MASTER EN SCIENCES INFORMATIQUES

Contribution à la réalisation d'un atelier de conception de bases de données

Rase, Bruno

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'informatique

Rue Grandgagnage, 21
B-5000 NAMUR (Belgium)

Contribution à la réalisation d'un atelier de conception de bases de données

Mémoire présenté en vue de l'obtention
du diplôme de Licencié et Maître en Informatique

Bruno RASE

Promoteur : J-L. Hainaut

Année Académique 1991 - 1992

Je tiens tout d'abord à exprimer ma profonde gratitude à mon promoteur, Monsieur Jean-Luc Hainaut, pour les conseils et les encouragements qu'il m'a apportés tout au long de la réalisation de ce travail.

J'exprime également toute ma reconnaissance à Messieurs Paul-André Brès et Jean-Paul Néron ainsi qu'à tous les employés de la société CONCIS à Paris, pour m'avoir permis d'effectuer, dans les meilleures conditions, un stage dans leur entreprise.

J'adresse aussi mes remerciements à Messieurs Bernard Decuyper et Olivier Marchand sans qui la partie implémentation de mon mémoire n'aurait pas été possible.

Enfin, je remercie toutes les personnes qui ont participé de près ou de loin à l'achèvement de ce travail.

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue Grandgagnage 21, B-5000 NAMUR
Tél. 081-72.49.83
Télex 59.222 Fac.Nam.B
Téléfax 081-72.49.67

Contribution à la réalisation d'un atelier de conception de bases de données

Bruno Rase

Résumé

De nos jours, la conception de bases de données efficaces et opérationnelles demeure un processus complexe qui peut être assisté par un outil logiciel. Ce travail est une participation à la conception et à la réalisation d'un atelier logiciel d'aide à la conception de bases de données. L'analyse de la base des spécifications de l'atelier a été faite selon l'approche orientée objets, un langage de programmation orienté objets constituant le support de l'implémentation. La méthodologie de conception de bases de données et le modèle de spécification supporté par l'atelier sont présentés. Des transformations sur ce modèle de spécification ont été développées. Ces transformations sont de trois types : élémentaires, globales ou selon un modèle. La transformation vers le modèle relationnel et la génération des descriptions exécutables vers une base de données relationnelle ont également été implémentés.

Abstract

Nowadays, the design of efficient and operational databases is a complex process. This work is a participation to the conception and the implementation of a database CASE tool. Analysis of the specification database is undertaken according to the object oriented approach, the implementation is supported by a object oriented programming language. A database design process and the specification model of the tool are presented. Transformations on that specification model have been developped. These transformations are of three kinds : elementary, global or model-driven. The transformation towards the relationnal model and the generation into the data description language of a relationnal database have also been implemented.

Mémoire de licence et maîtrise en Informatique
Septembre 1992
Promoteur : J-L. Hainaut

Table des matières

Introduction

1. Cadre général de travail

L'élaboration d'une base de données, prise dans le cadre du développement d'une application informatique, constitue un processus complexe. Pour pouvoir obtenir une base de données qui soit une représentation fidèle du système et qui soit opérationnelle et efficace, il est nécessaire de suivre une démarche de conception. Cette démarche, pour être totalement efficace, doit être accompagnée d'un outil logiciel d'aide à la conception, assurant la maîtrise aux différents niveaux du processus de conception.

Ce mémoire constitue une participation à la conception et à l'implémentation d'un atelier de conception de bases de données. Il concerne plus particulièrement la réalisation des outils de transformation et de génération.

2. Originalité de ce travail

L'originalité de ce travail réside dans le choix de l'environnement de développement pour l'atelier de conception. En effet, nous avons choisi comme langage de programmation, le langage orienté objet C++. La méthode de développement orientée objet semble être plus souple que l'approche purement fonctionnelle traditionnellement utilisée.

3. Organisation du travail

Le premier chapitre présente la démarche de conception d'une base de données telle qu'elle est envisagée avec l'apport d'un outil d'aide à la conception. Cette démarche est inspirée de [BOPI-89] et [HAIN-86].

Une démarche de conception n'est pas envisagée sans l'apport d'un modèle de données, la description de ce modèle de données est le sujet du deuxième chapitre de ce travail. Le modèle de données supporté par l'atelier constitue l'ensemble des données qui sont manipulées par le programme. Ces données sont en fait des méta-données car elles décrivent un autre ensemble de données (la base de données en cours de conception). Le schéma décrivant ces données est appelé méta-schéma.

Le méta-schéma a été spécifié dans une perspective d'implémentation orientée objet, certains rappels concernant la programmation orientée objet sont donc fait et des techniques d'implémentation du schéma sont proposées.

Le troisième chapitre aborde le problème des transformations de schéma dans le cadre du développement d'une base de données. Ces transformations constituent l'étape essentielle du processus de conception et peuvent être automatisée pour aider le concepteur. Les

transformations présentées ne sont qu'un sous-ensemble des transformations possibles sur un schéma.

La dernière étape dans la conception d'une base de données consiste à écrire la description de la base de données dans le langage du SGBD choisi comme support de l'application. Cette étape peut également être automatisée et le texte exécutable peut être généré directement à partir des spécifications introduites dans l'atelier. Le quatrième chapitre du travail aborde le problème de la génération vers un SGBD relationnel.

Chapitre 1 : La conception des bases de données

Introduction

Ce premier chapitre aborde le problème de la conception des bases de données.

Dans un premier temps, un rappel concernant les **méthodologies de conception de bases de données** est fait. Cette section nous permet de préciser notre point de vue en matière de méthode de conception.

Ce rappel fait, nous exprimerons le rôle des **ateliers logiciels** dans la démarche de conception de la base de données.

1. Méthodologie de conception de bases de données

Cette section décrit dans un premier temps le rôle de la base de données dans un système d'information et présente l'idée de méthodologie de conception. Elle décrit ensuite les différentes étapes de la démarche de conception qui sont l'analyse conceptuelle, la conception logique et la conception physique.

1.1 Le système d'information et la base de données

Le modèle général [BOPI-89] d'un système d'information est présenté à la figure 1.1. On considère qu'un traitement reçoit des données, fournies par son environnement (c-à-d un utilisateur ou une autre application), les traite selon ses spécifications, et restitue des résultat à son environnement. Ce traitement consomme des ressources et interagit avec la base de données.

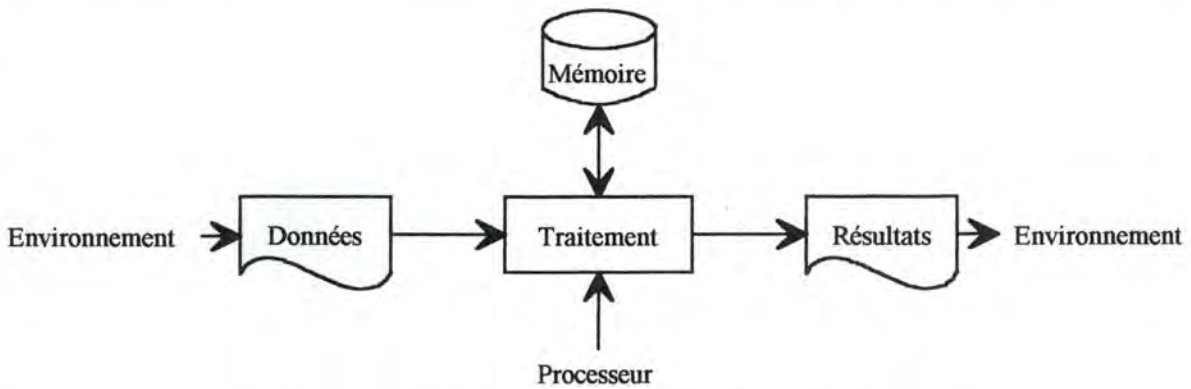


Figure 1.1 : Schéma du modèle général d'un système d'information

Précisons le rôle de la base de données. Les applications informatiques ont pour but de fournir des traitements automatisés qui peuvent avoir une très grande importance pour l'entreprise. La plupart de ces applications doivent interagir avec la base de données, qui est la *mémoire* du sytème d'information. La base de donnée est une collection de données représentant des informations du monde réel. Une base de données doit être :

- une représentation fidèle de la réalité,
- un serveur de données efficace.

1.2 Méthodologie de conception de bases de données

La structure de la base de données doit être connue de l'ordinateur. La vue globale de la base de données, c-à-d son *schéma*, est spécifiée selon un formalisme de structuration et de

description de la base de données, appelé *modèle de données*. Le processus de construction de la base de données est appelé *conception de la base de données*.

Il est actuellement reconnu que les systèmes d'information (et donc les bases de données) ne sont pas construits de façon arbitraire mais plutôt selon une méthodologie. Toute méthodologie est basée sur un ensemble de modèles, et propose différentes étapes et règles.

1.2.1 Les étapes de la conception

Le problème essentiel de la modélisation d'un sous-ensemble du monde réel est la différence entre la perception humaine de la structure de l'entreprise et les besoins de l'ordinateur pour organiser les données d'une façon particulière, avec des contraintes de stockage et de performance.

Ce problème a conduit à une modélisation de la base de données en niveaux. Les étapes de modélisation que nous considérons dans le cadre de ce travail, illustrées à la figure 1.2, sont les suivantes [HAIN-89] :

- L'étape *d'analyse conceptuelle* produit un schéma conceptuel décrivant les structures sémantiques que la base de données va contenir afin de représenter le *réel perçu*;
- L'étape *de conception logique* produit un schéma, qui doit exprimer toute la sémantique du schéma conceptuel, décrire les mécanismes d'accès nécessaires aux algorithmes de l'application et être conforme à un SGBD cible;
- L'étape *de conception physique* produit une description physique de la base de données (texte exécutable) qui doit être correcte, efficace et interprétable par le SGBD cible.

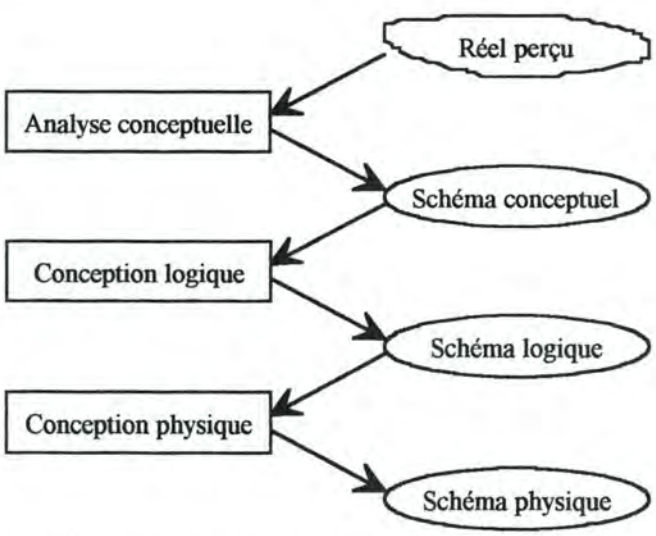


Figure 1.2 : Schéma général de la démarche de conception d'une base de données

1.2.2 Les modèles de données

Dans la plupart des méthodologies, à chaque étape de la démarche correspond un modèle de données différent. Par exemple, les spécifications conceptuelles sont écrites avec le modèle Entité/Association, les spécifications logiques utilisent un modèle binaire et le schéma physique est un schéma relationnel.

Certaines méthodologies passent directement du modèle conceptuel au modèle du SGBD qui est utilisé pour les étapes logiques et physiques, tandis que d'autres utilisent un modèle intermédiaire pour l'étape de conception logique [HAIN-86].

La démarche de conception décrite ci-dessous est basée sur un modèle de représentation unique qui permet d'exprimer la structure de la base de données à toutes les étapes de la démarche. Ce modèle est présenté au chapitre 2.

1.2.3 Les transformations

Quelle que soit la méthodologie de conception utilisée, elle est basée sur des principes de *transformations* de schéma qui permettent de passer d'un modèle de représentation à l'autre ou, si le modèle est unique, d'une étape de la démarche à l'autre.

Dans le cadre du modèle de spécification choisi, ces transformations ont trois objectifs fondamentaux : l'élimination de structures sémantiques avancées, l'élimination d'anomalies de représentation et l'élimination de constructions non conformes à un SGBD déterminé.

Un sous-ensemble des transformations sur ce modèle est décrit au chapitre 3.

1.3 L'analyse conceptuelle

1.3.1 Le réel perçu et la base de données

Le réel perçu représente un domaine de la réalité qui est pertinent pour l'application et dont on désire représenter les aspects statiques au moyen d'une base de données. Le réel perçu contient des composants individuels (les objets) qui sont en relation les uns avec les autres, et qui sont caractérisés par des propriétés.

La base de données (une partie du système d'information) peut être définie comme une représentation formelle du réel perçu.

Nous distinguons deux parties dans la base de données :

- Le *schéma conceptuel* est le résultat de la formalisation du réel perçu;
- Le *schéma des occurrences* contient la représentation des faits réels.

1.3.2 Le schéma conceptuel

Le schéma conceptuel décrit les structures sémantiques des données. On suppose qu'une telle description est indépendante de la notion d'outil informatique et compréhensible par tout le monde (utilisateurs, concepteurs et programmeurs). Ses objectifs sont :

- d'aider l'utilisateur et le concepteur à exprimer la signification des données, c'est-à dire l'information pertinente pour l'application,
- d'aider le progammeur du système à organiser les données correctement selon leur sémantique,
- de donner aux utilisateurs une définition précise des informations qu'ils manipulent, et leur condition d'utilisation.

1.3.3 Le modèle conceptuel

Le but des modèles conceptuels de données est de permettre au concepteur de construire plus facilement des structures "naturelles" et de permettre à l'utilisateur de manipuler des concepts qui lui sont déjà familier. Plus un modèle est capable de refléter le processus d'abstraction, plus il sera proche de la façon de voir de l'utilisateur et plus facile il sera à utiliser.

Le modèle le plus répandu pour la description conceptuelle des bases de données est le modèle Entité-Association [BOPI-89].

1.3.4 La démarche d'analyse conceptuelle

L'analyse conceptuelle a pour but de produire un schéma conceptuel qui spécifie les structures sémantiques que la base de données va contenir afin de représenter le réel perçu.

La démarche, illustrée à la figure 1.3, se présente comme suit. Le réel perçu est décomposé en sous-systèmes homogènes du point de vue de l'organisation. Suite à l'analyse des besoins relatifs à chaque sous-système, un schema conceptuel E-A est construit pour chacun d'entre eux. Ces sous-schémas sont normalisés pour éliminer les constructions conduisant à des anomalies de représentation. Tous les schémas sont alors intégrés pour former un schéma conceptuel global qui est validé par les utilisateurs.

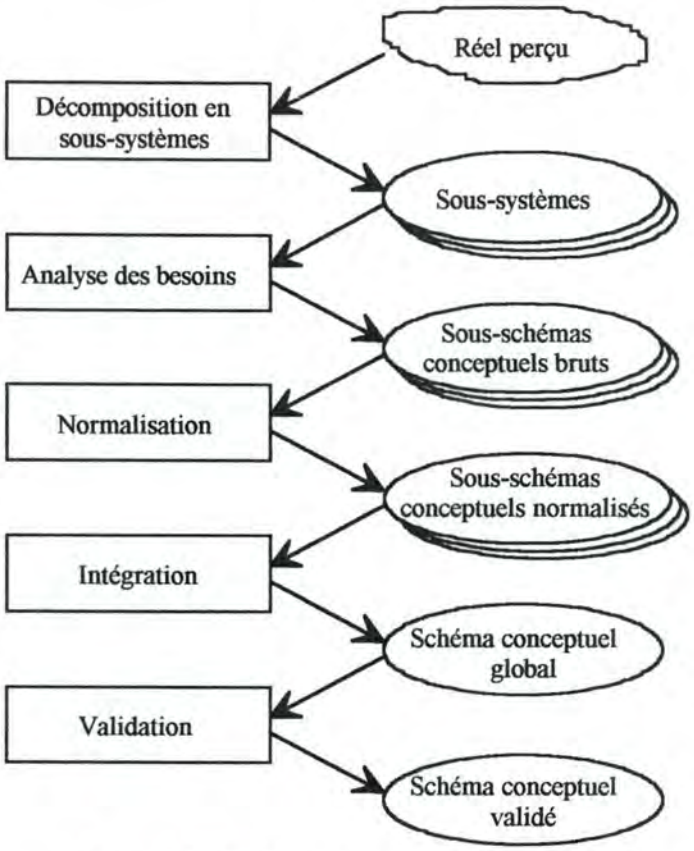


Figure 1.3 : La démarche d'analyse conceptuelle

1.4 La conception logique

1.4.1 Le modèle logique

Le modèle logique est constitué d'une part d'un sous-ensemble des structures E-A de base, et d'autre part de structures d'accès qui définissent des moyens d'accéder aux données (chemins d'accès et clés d'accès). Le sous-ensemble du modèle E-A est constitué des concepts de types d'entités, d'attributs, de types d'associations mais restreints aux associations de degré 2 (binaires) et sans attributs, et des contraintes d'intégrité.

1.4.2 La démarche de conception logique

La conception logique a pour but de produire un schéma qui reprenne toute la sémantique exprimée dans le schéma conceptuel. Il doit en plus décrire les besoins des applications en terme d'accès aux données et être conforme à un SGBD.

La démarche, appliquée à la conception d'un schéma relationnel, se présente comme suit (voir figure 1.4). Le schéma conceptuel validé doit être transformé en un schéma logique initial par élimination des structures E-A étendues et par élimination des types d'associations complexes[1]. On analyse ensuite les besoins en accès des applications et on produit un schéma

[1] Voir le §1.4 du chapitre 2 pour la définition d'un type d'association complexe.

des accès nécessaires exprimé dans le modèle E-A logique. Il reste alors à transformer ce schéma de manière à le rendre conforme au modèle du SGBD.

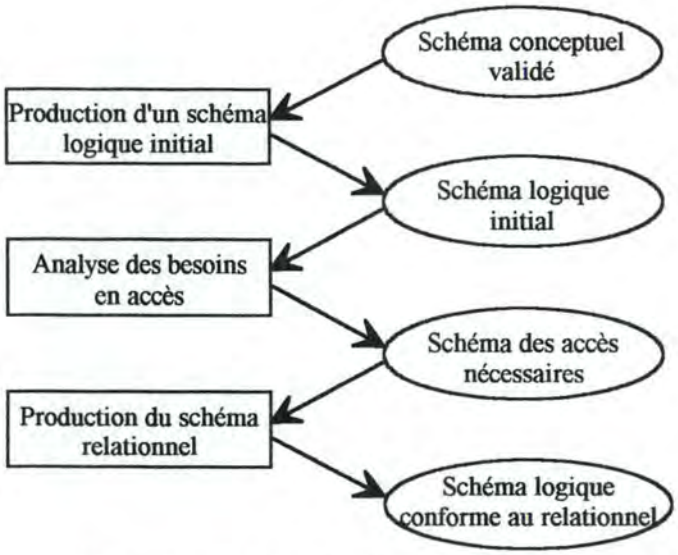


Figure 1.4 : La démarche de conception logique

1.5 La conception physique

Cette étape doit produire une base de données opérationnelle efficace.

Le schéma logique conforme permet de produire un texte de définition de la base de données utilisable par les programmeurs et donc directement exécutable. Si on applique la démarche à la production d'une base de données relationnelle (Ex :SQL), ce texte reprend d'un part la définition de la base et d'autre part la définition des index exprimant les clés d'accès spécifiées dans le schéma logique conforme. D'autres parmètres techniques peuvent ensuite être déterminé en fonction du profil des applications caractéristiques du SGBD (fréquence, mode d'exécution, temps de réponse,...).

2. Les ateliers de conception de bases de données

La démarche de conception peut en partie être assistée par un outil logiciel. Cette section présente dans un premier temps la notion d'outil CASE. Elle explique ensuite brièvement les fonctions et l'architecture d'un atelier de conception de bases de données.

2.1 Les outils CASE

Face à la relative complexité des systèmes d'information et à la difficultés de les concevoir, seul l'emploi d'outils automatisés peut aider le concepteur à vérifier que le schéma conceptuel qu'il construit est correct.

Les environnements logiciels d'aide au développement des systèmes d'information, présentés actuellement sous le nom de **C**omputer **A**ided **S**oftware **E**ngineering (CASE), fournissent un ensemble d'outils de deux types : actifs et passifs. Les outils *passifs* (ou front-end tools) sont des outils qui seront utilisés durant les phases de spécification, d'analyse et de design, ce sont généralement des éditeurs graphiques ou textuels et des générateurs de rapports documentaires. Les outils actifs (ou back-end tools) sont utilisés durant les phases d'implémentation et de test, ce sont des outils de prototypages, de simulation, des générateurs de code,etc...

A l'heure actuelle de plus en plus *d'outils d'aide à la conception de bases de données* apparaîssent. Ils constituent soit une variante d'un outil CASE, soit un composant de celui-ci. La démarche de conception décrite à la section précédente peut être en partie automatisée par un de ces outils.

2.2 Fonctions d'un outil de conception de bases de données

La fonction principale d'un outil de conception est de gérer la documentation concernant une base de données en projet. Cette fonction se décompose en sous-fonctions de saisie, de consultation et de modification des spécifications, ainsi que de production de documentation sur support externe. Cette fonction peut être couplée à des fonctions de transformation et de validation de spécifications, ainsi que de génération de descriptions destinées à un SGBD.

2.2.1 Saisie, consultation et mise à jour des spécifications

La saisie des spécifications peut se faire via un langage de spécification, ou bien grâce à un éditeur syntaxique ou graphique. Elle peut aussi se faire par analyse de descriptions exécutables (reverse engineering).

La consultation des spécifications peut se faire sous forme de texte ou de graphique, et grâce à un mécanisme de sélection et de navigation au travers des spécifications.

La modification des spécifications se fait également grâce un éditeur syntaxique ou graphique. Elle peut être assistée par un contrôle de cohérence en temps réel ou différé, et par des mécanismes de reprises arrière et avant (undo, redo).

2.2.2 Production de documentation

La production de documentation externe se fait sur un support externe (généralement papier) et est sous forme graphique ou textuelle. Elle peut être paramétrable.

2.2.3 Transformation et validation des spécifications

Les transformations des spécifications (voir chapitre 3) sont ponctuelles ou globales manuelles ou automatiques et peuvent être assistées par un mécanisme de reprise arrière et avant. Les modifications qu'entraînent ces transformations doivent avoir des répercussions au travers des spécifications dérivées.

La validation des spécifications peut être de trois types : une validation conceptuelle (syntaxique, cohérence et complétude, normalisation), une validation logique (conformité à un SGBD) ou une validation physique (conformité à un SGBD, complétude et cohérence des paramètres).

2.2.4 Production de descriptions exécutables

La production de descriptions exécutables consiste à générer partiellement ou complètement un texte de description de la base de données dans le langage de description de données (DDL) du SGBD cible choisi (voir chapitre 4).

2.3 Architecture type d'un atelier de conception de bases de données

Comme nous l'avons déjà dit, un outil de conception de bases de données est soit une variante d'un outil CASE, soit un composant de celui-ci. Dans les 2 cas, il s'articule autour d'une base de données qui contient la description des schémas relatifs au différentes étapes de la conception. Cette base de données est généralement appelée *base des spécifications*.

L'architecture type exposée dans [HAIN-89] peut être présentée comme suit. Les différentes fonctions décrites ci-dessus sont prises en charge par des processeurs différents, indépendants les uns des autre et travaillant tous sur le contenu de la base des spécifications. Toutes ces fonctions peuvent être exécutées dans un ordre indéterminé, ce qui permet à l'outil d'être relativement neutre par rapport aux différentes étapes de la démarche de conception. Ce type d'architecture permet également un extensibilité aisée des fonctionnalités.

Chapitre 2 : Schéma conceptuel orienté objet d'un atelier de conception de bases de données

Introduction

Ce chapitre est destiné à décrire le méta-schéma de l'atelier de conception de bases de données.

La première section du chapitre, **le modèle de spécification**, décrit le modèle de données qui sera supporté par l'atelier logiciel.

La seconde section concerne **la programmation par objet**, elle présente certains concepts de la programmation orientée objet et décrit leur incorporation dans le language C++.

La troisième section, **description de la base des spécifications**, est une description du méta-schéma conceptuel de l'atelier logiciel

La dernière section, **implémentation du schéma**, présente les techniques d'implémentation utilisées pour le développement de la base des spécifications de l'atelier.

1. Le modèle de spécification

Le modèle de spécification de l'atelier logiciel est basé sur une extension du modèle Entité-Association. Le modèle E-A de base, qui peut être étudié plus en détail dans [BOPI-89], reprend le notions de types d'entités, types d'association et rôles, attributs, identifiants et contraintes d'intégrité. Nous y avons ajouté quelques extensions telles que les rôles multi-domaines, les structures de Généralisation/Spécialisation, les groupes et les domaines de valeurs. Ce modèle de spécification permet de couvrir toutes les étapes de la démarche de conception d'une base de données.

Cette section décrit les concepts suivants : système, schéma, type d'entité, type d'association et rôle, attribut, groupe, domaine, valeur, contrainte et la notion de généralisation/spécialisation.

1.1 Le système

La notion de système permet de regrouper un ensemble de schémas (ou bases de données) d'un même système d'information. Les différents schémas d'un système peuvent décrire des bases de données différentes ou bien des bases de données identiques mais à des niveaux de description différents.

1.2 Le schéma

Un schéma permet de représenter la description d'une base de donnée particulière à n'importe quelle étape de la démarche de conception (schéma conceptuel, logique ou physique).

Il contient un certain nombre de types d'entités, de types d'associations et d'attributs. Un schéma a un nom, un nom court, un numéro de version, un niveau et un status; il peut également être associé à d'autres schémas, par exemple : le schéma X est le schéma physique correspondant au schéma conceptuel Y.

1.3 Les types d'entités

"*Une entité est une chose concrète ou abstraite appartenant au réel perçu à propos de laquelle on veut enregistrer des informations.*" [BOPI-89]

Toutes les entités du réel perçu sont distinctes mais elles peuvent être regroupées et former une classe. Un type d'entité représente un classe d'entités du réel perçu. Le modèle E-A de base fait l'hypothèse que toute entité appartient à une et une seule classe mais dans notre modèle, suite

à l'introduction de la notion de Généralisation/Spécialisation[1], on autorise qu'une entité appartienne à plus d'un type d'entité.

Dans un système de gestion de bases de données, une entité est la représentation ou l'image d'un objet du réel perçu. Dans l'atelier logiciel, un type d'entité peut aussi bien décrire des entités conceptuelles que des objets techniques (tels que enregistrements, tables,...).

Un type d'entité contient des attributs et des groupes, joue des rôles dans des types d'associations et appartient à un schéma. Il possède également un nom qui l'identifie dans un schéma et un nom court.

Un type d'entité peut être représenté graphiquement par un rectangle dont l'entête contient le nom du type d'entité. Par exemple (figure 2.1), les types d'entités AUTEUR et LIVRE.

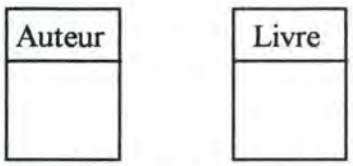


Figure 2.1 : Exemple de types d'entités

1.4 Les types d'associations et les rôles

"*Une association est définie par une correspondance entre deux ou plusieurs entités (non nécessairement distinctes) où chacune assume un rôle donné.*" [BOPI-89]

Toute association appartient à une et une seule classe appelée type d'association.

Un type d'association est défini entre des types d'entités; chaque type d'entité jouant un rôle dans le type d'association. Au moins deux rôles sont définis pour chaque type d'association; le nombre de rôles définis est appelé *degré* du type d'association. Un type d'entité peut jouer plusieurs rôles dans un seul type d'association.

Chaque rôle est caractérisé par une contrainte de cardinalité. Cette contrainte est définie par un couple d'entier I-J ($I \geq 0$, $J \geq 1$, $J \geq I$) , qui indique qu'une entité peut jouer de I à J fois ce rôle; I et J sont respectivement appelés cardinalité minimum et cardinalité maximum du rôle. Si J est un nombre arbitrairement grand, il est représenté par la lettre N.

On peut distinguer deux sortes de types d'associations. Les types d'assocations *complexes* sont de degré supérieur à 2, ou dotés d'attributs. Les types d'associations *fonctionnels* sont de degré 2, sans attributs et dotés d'au moins un rôle de cardinalité 0-1 ou 1-1.

[1] Voir §1.10

Un type d'association a des rôles, des attributs, des groupes et appartient à un schéma. Il possède également un nom qui l'identifie dans un schéma et un nom court.

Un type d'association peut être représenté graphiquement par une ellipse avec un en-tête qui contient le nom du type d'association et des liens qui le relient aux types d'entités. Ces liens sont étiquetés de leur cardinalité et éventuellement du nom du rôle joué par le type d'entité dans le type d'association. Par exemple (figure 2.2), le type d'association ECRITURE entre AUTEUR et LIVRE qui représente le fait qu'un auteur écrit des livres.

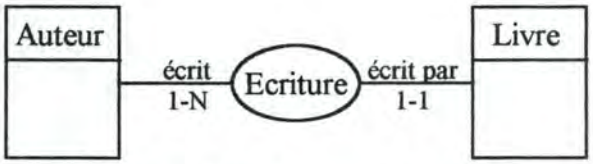


Figure 2.2 : Exemple de type d'association

On peut également admettre qu'un rôle soit joué par une entité choisie parmi plus d'un type. On parle alors de rôles multi-domaines. On peut voir un exemple sur la figure 2.3 qui montre qu'une PERSONNE peut posséder une VOITURE, un CAMION ou une MOTO.

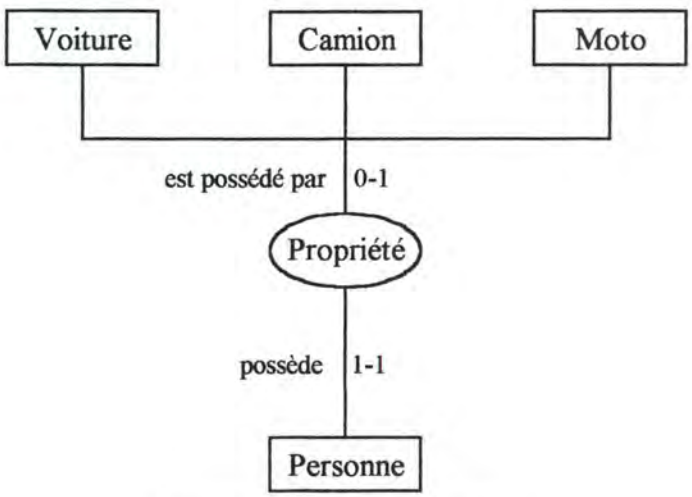


Figure 2.3 : Rôle multi-domaines

1.5 Les attributs

"*Attribut : caractéristique ou qualité d'une entité ou d'une association. Il peut prendre un(e) ou plusieurs valeurs ou groupes de valeurs.*"[BOPI-89]

Un attribut appartient à un type d'entité ou un type d'association et il peut prendre 0, 1 ou plusieurs valeurs pour chaque instance du type d'entité ou du type d'association.

Chaque attribut est caractérisé par une contrainte de cardinalité. Cette contrainte est définie par un couple d'entier I-J ($I \geq 0$, $J \geq 1$, $J \geq I$) , qui indique que le nombre de valeurs de l'attribut

associées à chaque occurence du type d'entité ou d'association est entre I et J. Un attribut peut être *facultatif* (I = 0) ou *obligatoire* (I > 0), *simple* (J = 1) ou *répétitif* (J > 1).

Un attribut qui est constitué d'autres attributs est dit *décomposable*, sinon il est *élémentaire*. Les composants d'un attribut peuvent eux-mêmes être élémentaires ou décomposables. Un attribut qui appartient directement à un type d'entité ou un type d'association est dit du *premier niveau*.

Un attribut possède un nom qui l'identifie parmi les attributs du type d'entité ou du type d'association (pour les attributs du premier niveau) et parmi les autres composants d'un attribut décomposable. Un attribut élémentaire peut être associé à un domaine de valeurs (ou type de valeurs). Un domaine de valeurs représente un ensemble de valeurs de même structure. Ce domaine peut être prédéfinis (Ex : caractère, entier, booléen, date,...) ou défini par l'utilisateur du modèle[2]. Un attribut élémentaire est également caractérisé par une longueur et éventuellement par un nombre de décimales.

Graphiquement, on met l'attribut dans la partie inférieure du rectangle ou de l'ellipse représentant le type d'entité ou le type d'association auquel il est associé. Si l'attribut est répétitif ou facultatif, sa cardinalité est indiquée entre crochet. La structure d'un attribut décomposable est indiquée par une indentation à chaque niveau de décomposition. La figure 2.4 présente les attributs du type d'entité PERSONNE.

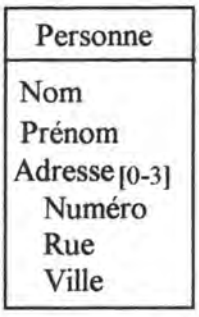


Figure 2.4 : Exemple d'attributs

1.6 Les groupes

Un groupe est une construction qui permet de représenter différentes notions telles que l'identification d'un type d'entité ou d'un type d'association, l'intégrité référentielle, les clés d'accès, etc.

Un groupe est une collection d'attributs et/ou de rôles et/ou d'autres groupes, associé à un type d'entité, un type d'association ou un attribut.

[2] Voir §1.8

Les attributs appartenant à un groupe d'un objet doivent être des attributs de cet objet et les rôles doivent être des rôles *voisins* de l'objet. Un rôle voisin d'un type d'association est un des rôles définis sur le type d'association, tandis qu'un rôle voisin d'un type d'entité **TE** est un des rôles que jouent d'autres types d'entités dans les types d'associations dans lesquelles **TE** apparaît dans un rôle déterminé (on se limite en général aux types d'associations binaires). Les groupes appartenant à un autre groupe peuvent être des groupes associés à un autre objet. Cela permet d'inclure dans un groupe des attributs ou des rôles "éloignés", c'est-à-dire appartenant à d'autres types d'entités. Par exemple (figure 2.5), le groupe identifiant du type d'entité BUREAU est composé de son NUMÉRO et d'un groupe du type d'entité ETAGE composé du rôle joué par le type d'entité IMMEUBLE dans le type d'association CONTIENT. Cela signifie qu'un BUREAU est identifié par son numéro dans un IMMEUBLE.

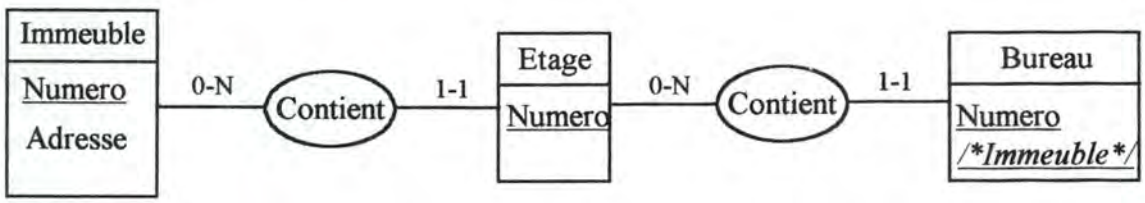
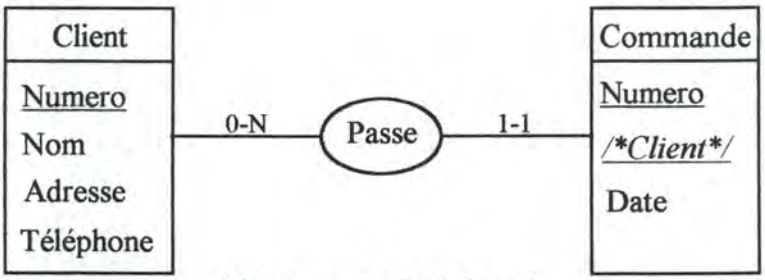


Figure 2.5 : Exemple de composition de groupe

Un groupe joue une ou plusieurs fonctions spécifiques pour l'object auquel il est attaché :

- Il peut être un *identifiant* : c'est-à-dire qu'à chaque combinaison de valeurs prises par ce groupe correspond au plus une occurrence du type d'entité ou du type d'association auquel il est associé. Un objet peut contenir plusieurs groupes identifiants. Les groupes identifiants sont de deux types : *primaire* ou *secondaire*. L'identifiant primaire représente l'identifiant principal de l'objet auquel il appartient, ses composants doivent être constants et obligatoires. Il ne peut y avoir qu'un seul identifiant primaire par objet, les autres identifiants sont des identifiants secondaires. Seul le groupe identifiant primaire peut être représenté graphiquement en soulignant les attributs qui le composent et s'il s'agit de rôles, en reprenant leur nom dans la liste des attributs, en le soulignant et en l'entourant des caractère "/*" et "*/"[3]. Les identifiants secondaires doivent être décrit de manière textuelle. Par exemple (figure 2.6), un CLIENT a comme identifiant primaire son NUMERO et comme identifiant secondaire sont TÉLÉPHONE; et une COMMANDE est identifiée par son NUMERO et le CLIENT qui passe cette commande.

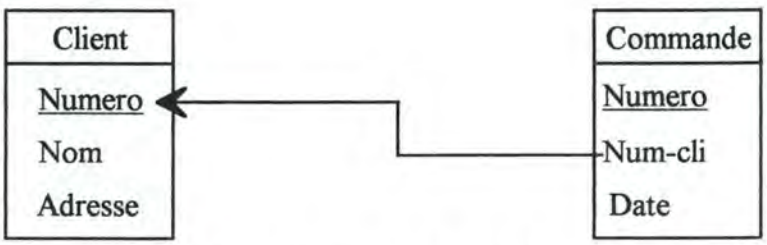
[3] Cette représentation est arbitraire, d'autre représentation peuvent être choisie (Ex : soulignement du nom du rôle identifiant,...).



Id(Client) = {Téléphone}

Figure 2.6 : Exemple de groupes identifiants

• Il peut être un groupe de *référence* : c'est-à-dire un groupe composé exclusivement d'attributs et qui fait référence à un autre groupe appelé groupe référencé. Cette notion de groupe de référence doit être associée à la notion de contrainte référentielle[4] pour représenter le mécanisme d'intégrité référentielle. Une référence peut être une inclusion ou une égalité, c'est-à-dire que l'ensemble des valeurs du groupe de référence peut être inclus ou égal à l'ensemble des valeur du groupe référencé. Par exemple (figure 2.7), le groupe composé de l'attribut NUM-CLI du type d'entité COMMANDE est inclus dans le groupe identifiant du type d'entité CLIENT.

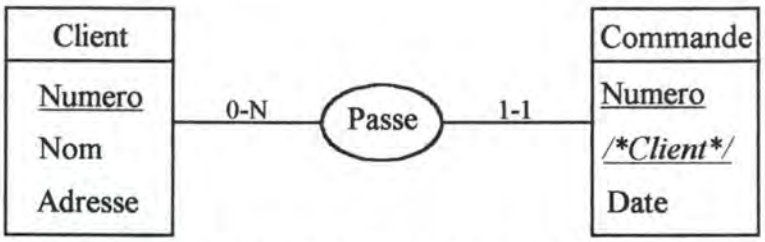


tout COMMANDE.Num-cli est un CLIENT.Numero

Figure 2.7 : Exemple de contrainte référentielle

• Il peut être, au niveau technique, une *clé d'accès* : c'est-à-dire un groupe tel qu'il existe un mécanisme permettant d'accéder successivement aux entités auxquelles est associé une valeur de cette clé. Par exemple (figure 2.8), NUMERO et l'entité CLIENT qui passe la commande constitue une clé d'accès à COMMANDE (c-à-d pour un CLIENT donné et une valeur de NUMERO on peut accéder rapidement aux entités COMMANDE concernées).

[4] Voir §1.10



key(COMMANDE) = { NUMERO,CLIENT}

Figure 2.8 : Exemple de clé d'accès

Ces trois fonctions peuvent être cumulées, on peut dès lors définir un groupe comme étant un identifiant et une clé d'accès.

1.7 Les domaines

Un domaine permet de définir et de nommer des contraintes de valeurs (format et groupe de valeurs) associées à des attributs de types d'entités et de types d'associations. Les domaines permettent d'élargir indéfiniment la listes des types prédéfinis (numérique, alphanumérique, date, booléen). Un même domaine peut être utilisé par plusieurs attributs et à un attribut correspond un et un seul domaine.

Comme un attribut, un domaine peut être *simple* ou *répétitif* et *élémentaire* ou *décomposable*.

Par exemple, le domaine décomposable *Adresse*, composés des domaines simples *Numéro*, *Rue*, *Code-postal* et *Ville*, où *Numéro* est de type numérique et de longueur 2, *Rue* est de type caractère et de longueur 20, *Code-postal* est de type numérique et de longueur 4 et *Ville* est de type caractère et de longeur 20. Les Attributs *Adresse-client* du type d'entité CLIENT et *Adresse-société* du type d'entité SOCIETE peuvent être associés tous les deux au domaine *Adresse*.

Les domaines nous mettent en présence d'un concept dynamique. En effet, le format du domaine et les contraintes de valeurs définies sur le domaine peuvent subir, à tout instant, des modifications et celles-ci sont supportées directement par l'ensemble des attributs associés au domaine.

1.8 Les Valeurs

Il est souvent utile d'associer à certains objets un ensemble de valeurs. Cet ensemble peut être soit une liste de valeurs que l'objet peut prendre (valeurs autorisées), soit une liste de valeurs que l'objet ne peut pas prendre (valeurs interdites). Cet ensemble peut également être représenté par un intervalle de valeur que l'objet peut prendre ou ne pas prendre. Les objets sur lesquels portent ces restrictions peuvent être des attributs ou des domaines élémentaires.

Par exemple, un attribut *sexe*, qui est défini de type alphanumérique et de longueur 1, peut avoir comme valeurs autorisées les lettres "M" pour masculin et "F" pour féminin.

1.9 Les contraintes

Les contraintes reprises dans cette section ne reprennent qu'une partie des contraintes d'intégrité décrites dans [BOPI-89] (les contraintes sur les rôles et sur les types d'associations) auxquelles nous avons ajouter d'autres contraintes (sur les groupes).

Contraintes sur les rôles :

• Inclusion de rôles : permet de représenter le fait que si une entité joue un rôle déterminé dans une association, alors cette entité doit également jouer un rôle précis dans une autre association. Par exemple (figure 2.9), une PERSONNE qui *loue* une VOITURE doit *posséder* un PERMIS DE CONDUIRE.

• Exclusion de rôles : permet de modéliser le fait que différents rôles, qui pourraient être joués par les mêmes occurences d'un type d'entité, sont mutuellement exclusifs.

• Egalité de rôles : permet de modéliser le fait que si une entité joue un rôle donné, alors elle doit également jouer un autre rôle, et inversément.

Les contraintes d'inclusion, d'exclusion et d'égalité d'association sont similaires aux contraintes sur les rôles excepté le fait que, par définition, elles concernent tous les rôles de l'association.

Les contraintes d'inclusion, d'exclusion ou d'égalité de groupes expriment le fait que l'ensemble des valeurs d'un groupe peut être inclus, égal ou différent de l'ensemble des valeurs d'un autre groupe. La contrainte référentielle exprime le fait qu'un groupe est une référence à un autre groupe; Cette contrainte permet d'exprimer l'intégrité référentielle dans le modèle relationnel.

Graphiquement les contraintes peuvent être représenté par un cercle dans lequel on inscrit un signe qui désigne le type de contrainte : la lettre "*I*" pour l'*inclusion*, le signe "*X*" ou "#" pour l'exclusion, etc. Ce cercle est relié aux objets sur lesquel la contrainte porte par des liens, éventuellement orientés (contrainte d'inclusion). Par exemple (figure 2.9), une contrainte d'inclusion entre les rôles *loue* et *possède*.

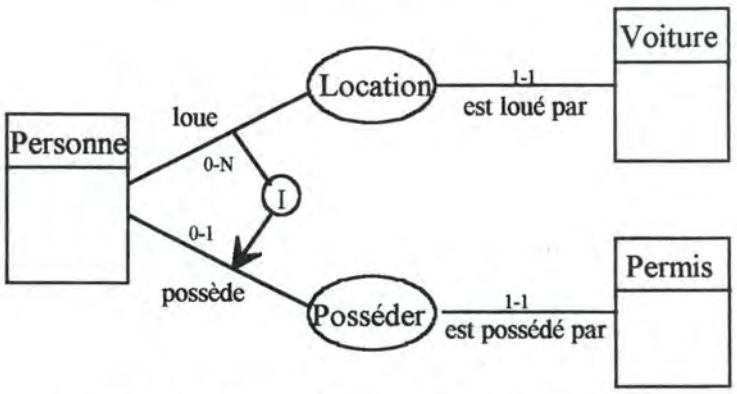


Figure 2.9 : Exemple de contrainte d'inclusion de rôles

1.10 La Généralisation/Spécialisation des types d'entités

Par le mécanisme de Généralisation/Spécialisation, on permet qu'une entité appartiennent à plus d'un type d'entité. On peut donc représenter par exemple qu'une même PERSONNE est également un EMPLOYE ou un OUVRIER, un HOMME ou une FEMME. Nous avons donc un objet générique, appelé également sur-type (PERSONNE), et des objets spécifiques, appelés sous-types (EMPLOYE, OUVRIER, HOMME et FEMME).

On définit la notion de sous-type comme suit : Si E_1 est un sous-type de E_2, alors toute occurrence de E_1 est aussi une occurrence de E_2, et cette occurrence hérite des propriétés de E_2, c'est-à-dire les attributs et associations auxquelles cette occurrence de E_1 participe.

A cette notion de généralisation/spécialisation, on peut adjoindre des contraintes d'intégrité portant sur les relations créées en spécialisant un type d'entité générique ou en généralisant un type d'entité spécifique. Il existe trois types de contraintes :

• *La disjonction* : Si une entité appartient à un type d'entité spécifique alors elle n'appartient à aucun autre type d'entité spécifique (dans l'exemple : un EMPLOYE ne peut être OUVRIER et réciproquement).

• *La couverture* : Si une entité appartient au type d'entité générique alors elle appartient à au moins un type d'entité spécifique (dans l'exemple : il n'existe pas de PERSONNE qui ne soit ni EMPLOYE, ni OUVRIER).

• *La partition* : Si une entité appartient au type d'entité générique alors elle appartient à un et un seul type d'entité spécifique (disjonction et couverture).

Graphiquement, on peut représenter la notion de généralisation/Spécialisation de la façon suivante :

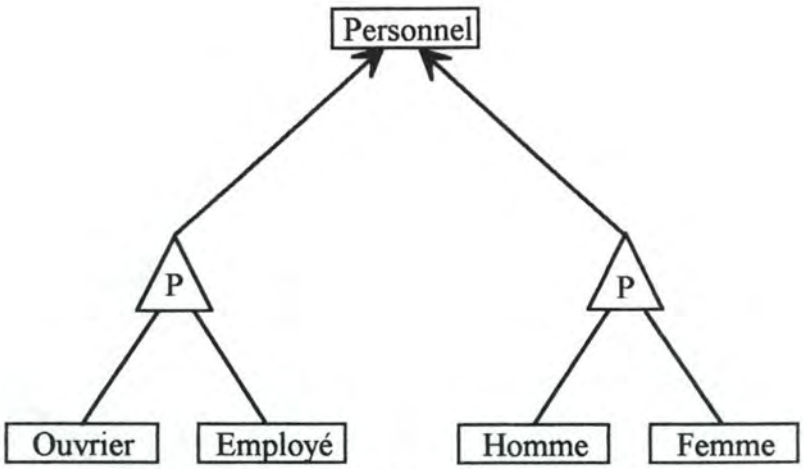


Figure 2.10 : Exemple de Généralisation/Spécialisation

Un membre du personnel est un ouvrier ou un employé et un homme ou un femme. Le triangle permet de représenter un ensemble d'entités spécifiques, il s'agit en quelque sorte d'un concept faisant "relais" entre le type d'entité générique et les types d'entités spécifiques. Les lettres "P", "D" et "C" indique s'il s'agit d'une partition, d'une disjonction ou d'une couverture.

2. La programmation par objets

Cette section est destinée à introduire les concepts propres aux langages orientés objets. La description porte principalement sur les notions qui ont été appliquées dans le cadre du développement de la base des spécifications et des modules de transformation et de génération.

2.1 Concepts fondamentaux de la programmation orientée objet

Le principe fondamental de la programmation par objet est qu'il faut structurer le logiciel autour des données plutôt qu'autour des actions. Dans un premier temps, nous définirons la notion d'objet, d'encapsulation, d'interface et de classe d'objets. Nous décriront ensuite les techniques d'héritage, de polymorphisme et de liaison dynamique.

2.1.1 L'objet

Les objets sont les entités de base des systèmes orientés objets. Ils représentent des choses concrètes ou abstraites du monde réel. Par exemple, le nombre π peut être représenté par un objet, ou encore le client *Dupont*.

2.1.2 L'encapsulation et l'interface

A chaque objet est associé un ensemble de fonctions, appelées méthodes, qui définissent des opérations applicables à l'objet. L'ensemble des méthodes est appelé **l'interface** de l'objet. Par exemple, pour le nombre π , l'interface associée à l'objet le représentant pourrait se composer des opérations algébriques (addition, soustraction,...); et pour le client *Dupont*, l'interface pourrait se composer des opérations de modification des caractéristiques d'un client (son adresse, son numéro de téléphone,...).

Dans chaque objet, il y a donc fusion des données permettant de représenter l'objet et des méthodes assurant la manipulation de ces données. Cette fusion de données et de méthodes est appelée **encapsulation**.

2.1.3 La classe

Les classes d'objets décrivent des ensembles de structures de données (objets) ayant des propriétés communes. Il ne faut pas confondre classe et objet : la différence est la même qu'entre un ensemble d'éléments, par exemple l'ensemble des nombres réels ou l'ensemble des clients de l'entreprise, et un élément particulier, par exemple le nombre π ou le client *Dupont*.

Les différentes opérations associées à l'interface d'un objet peuvent également être appliquées aux autres objets de la classe. Dès lors, tous les objets faisant partie de la même classe d'objets contiennent une information de même type et ont la même interface.

Chaque occurence d'une classe d'objets est une **instance** de cette classe. Par exemple, le nombre π et le client *Dupont* sont respectivement des instances de la classe des nombres réels et de la classe des clients.

2.1.4 L'héritage

Le mécanisme d'héritage permet de définir une nouvelle classe comme une extension ou une restriction d'une autre classe. Cela diminue fortement la quantité de code nécessaire pour définir une nouvelle classe d'objets et, de plus, un changement des propriétés d'une classe d'objets se répercute immédiatement sur toutes les classes héritant des propriétés de la classe modifiée. Cela facilite grandement la maintenance des applications développées à l'aide d'un langage orienté objets.

Quand une classe d'objets B hérite d'un autre classe d'objets A (voir figure 2.11), la classe d'objets A est appelée classe de base et la classe d'objets B est appelée classe dérivée. La classe d'objets B contient une partie dérivée constituée des propriétés héritées de la classe d'objets A et un partie incrémentale spécifique à la classe d'objets B. En fait, lorsqu'une classe d'objets B est dérivée d'un classe d'objets A, cette classe d'objets B hérite de toutes les propriétés de la classe d'objets A ce qui implique que tout objet appartenant à la classe d'objets B est également une instance de la classe d'objets A. On a donc une relation d'inclusion de la classe d'objets B dans la classe d'objets A.

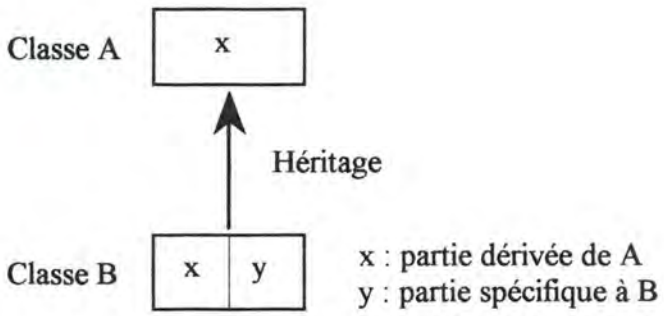


Figure 2.11 : Illustration du mécanisme d'héritage

On peut remarquer que selon le langage orienté objets utilisé, soit la partie dérivée de la classe d'objets B est identique à la classe d'objets A, soit la partie dérivée peut être une réimplémentation complète ou partielle de la classe d'objets A.

2.1.5 Le polymorphisme

Le polymorphisme est la faculté de prendre différentes formes. En programmation par objets, cela caractérise une entité qui fait référence au moment de l'exécution à des occurences de différentes classes. Une référence polymorphe est donc une référence à un objet qui peut appartenir à plusieurs classes d'objets différentes. L'ensemble des classes auxquel l'objet peut appartenir est déterminé à partir de la hiérarchie des classes d'objets du système.

Un référence polymorphe possède donc un type d'objet dynamique qui peut varier d'une instruction à l'autre durant l'exécution du programme, et un type d'objet statique qui est connu au moment de la déclaration de l'objet dans le programme et qui détermine l'ensemble des classes que la référence dans cet objet peut accepter au moment de l'exécution.

La relation d'héritage est fortement couplée avec la notion de polymorphisme. En effet, si la classe d'objets B hérite de la classe d'objets A, toute instance de la classe d'objets B fait également partie de la classe d'objet A, et partout dans le programme où une instance de la classe d'objet A est référencée, une instance de la classe d'objets B est permise.

2.1.6 La liaison dynamique

La liaison dynamique permet au moment de l'exécution de sélectionner la version de l'opération adaptée à une certaine occurence. Une opération peut donc avoir des versions différentes dans des classes différentes.

Par exemple, soit une référence polymorphe R qui a un type statique A et des types dynamiques A ou B. Supposons que A possède une procédure P et que cette procédure P à été redéfinie au niveau de la classe B, la procédure appelée varie suivant le type dynamique de la référence polymophe R. Ce choix se fait à l'exécution et est résolu grâce au concept de liaison dynamique.

2.2. Le langage orienté objet C++

Le langage de programmation C++ a été définit et implémenté en 1985 par Bjarne Stroustrup. Mis à part quelques détails mineurs, C++ est un sur-ensemble du langage C. Il a été conçu pour être un meilleur C, supporter les abstractions de données et supporter la programmation par objet. Le lecteur intéressé par une description complète du langage C++ peut consulter [STRO-92].

Cette section décrit la manière dont les concepts introduits dans la section 2.1 sont intégrés dans le langage C++. Les notions de classes, de classes dérivées, de constructeurs et destructeurs seront introduites.

2.1.1 définition des classes

La déclaration d'un classe d'objets en C++ revient à définir un nouveau type de données. Cette déclaration se fait à l'aide du mot réservé **class**. La structure d'une classe s'apparente fort à celle d'une structure de données (**struct**) pour ce qui est de la définition des données de la classe d'objets mais, à cette structure viennent s'ajouter les fonctions membres (méthodes) qui constituent l'interface de l'objet.

Exemple :

```
class person {
   int number;
   char* name;
public :
   char* get_name() { return name };
   void print();
};
```

Le label **public** sépare le corps de la classe en deux parties. Les noms de la première partie, *privée*, peuvent être uniquement utilisés par les fonctions membres. La seconde partie, *publique*, constitue l'interface des objets de cette classe. Les déclarations des fonctions sont celle définie par la syntaxe du langage C. Lors de l'implémentation des méthodes d'une classe d'objets, qui se fait en dehors de la déclaration de cette classe d'objets, il faut faire précéder le nom de la méthode par le nom de la classe d'objets à laquelle elle appartient.

Exemple :

```
void  person :: print()
{
   cout << name << " est une personne";
}
```

2.2.2 Les classes dérivées

La notion de classe dérivée est offerte pour exprimer des relations de hiérarchie. Les classes dérivées sont définies en indiquant après le nom de la classe, le nom de la classe d'objets de base de la classe déclarée.

Exemple :

```
class employee :: public person {
   char* department;
public :
   char* get_department() { return department};
   void print() { cout << name << " est un employé"; }
};
```

Le mot réservé **public** précédant le nom de la classe de base indique que les membres public de cette classe de base sont également des membres public de la classe dérivée.

La fonction *print()* de la classe *EMPLOYEE* est une redéfinition de la fonction *print()* de la classe *PERSON* adaptée aux objets de la classe. Dans ce cas, la partie dérivée de la classe d'objets *EMPLOYEE* est une redéfinition partielle de la classe d'objets *PERSON*.

2.2.3 Constructeurs et destructeurs

Les constructeurs et destructeurs sont des fonctions permettant de délimiter le cycle de vie d'une variable objet, instance d'une classe d'objets.

Une fonction membre de même nom que celui de la classe est appelée un **constructeur**; elle est utilisée pour construire des valeurs du type de sa classe. Elle sert à définir la façon dont l'objet sera initialisé lors de sa création. Si une classe possède un constructeur, chaque objet de cette classe sera initialisé avant toute utilisation de cet objet.

Un fonction membre d'une classe Cl et nommée ~Cl est appelée un destructeur; elle est utilisée pour détruire les valeurs de type Cl immédiatement avant que l'objet les contenant ne soit détruit.

Exemple :

```
class person {
   int number;
   char* name;
public :
   person (int num);
   ~person ();
   char* get_name() { return name };
   void print();
};
```

Ces fonctions ne sont pas appelées explicitement, elles sont appelées lors de la décaration d'une variable de classe (création statique) ou par les instructions **new** et **delete** (création dynamique). La création dynamique d'un objet d'une classe se fait grâce à l'instruction **new**, la fonction constructeur de la classe est alors exécutée. La fonction new renvoie l'adresse de l'objet créé et alloue la mémoire nécessaire. La destruction d'un objet créé de façon dynamique se fait grâce à l'instruction **delete**, qui désalloue automatiquement la mémoire.

Par exemple :

```
new person;
```

crée un objet de classe X et

```
delete person;
```

le détruit.

2.2.4 Déclaration d'une variable objet et appel des méthodes d'une variable objet

La déclaration d'une variable objet se fait tout naturellement en faisant précéder le nom de la variable du nom de la classe. La déclaration d'un pointeur sur une variable objet se fait grâce à l'opérateur *. Exemple :

```
person   p1;    // p1 est une variable de classe person
person   *ptr;  // ptr est un pointeur sur une variable de classe person
```

La technique d'appel des méthodes d'une variable objet particulière est dérivée directement de celle utilisée pour accéder aux différents membres d'une structure dans le langage C, on utilise les opérateur "." et "→". L'opérateur "." relie le nom d'une variable de classe à la méthode appelée et l'opérateur "→" relie le nom d'un pointeur sur une variable de classe à la méthode. Exemple :

```
p1.print();
ptr→print();
```

3. Description de la base des spécifications

3.1. Schéma Entité/Association de la base des spécifications

Cette section décrit le schéma conceptuel de la base de données des spécifications. On peut trouver une description complète du schéma dans [DEMA-92], nous ne faisons içi que présenter les objets du modèle de spécification décrit à la section 1 de ce chapitre.

La complexité du schéma a conduit à une présentation progressive sous forme de sous-schémas.

3.1.1 Architecture d'un système d'information

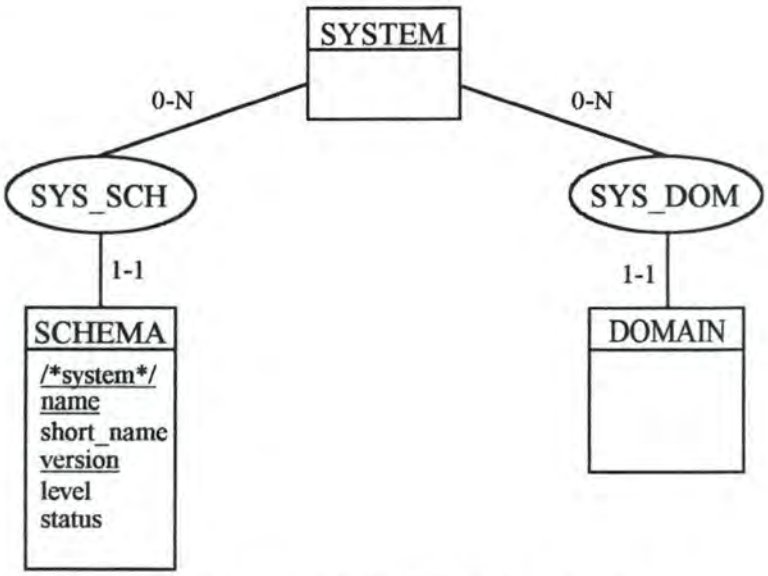


Figure 2.12 : Le type d'entité SYTEM et ses associations

La base de donnée contient la description d'un ou plusieurs systèmes d'informations (**SYSTEM**). Un **SYSTEM** possède une ou plusieurs descriptions de données (**SCHEMA**) et contient un certain nombre de domaines (**DOMAIN**) qui seront associés aux attributs du schéma.

Une description de données prend la forme d'un **SCHEMA** attaché à un **SYSTEM** (via **SYS-SCH**). Dans un **SYSTEM**, un **SCHEMA** est identifié par son nom (***name***) et son numéro de version (***version***), et il est caractérisé par un nom court (***short_name***), un niveau (***level***) et un status (***status***).

3.1.2 Structure de base d'un schéma

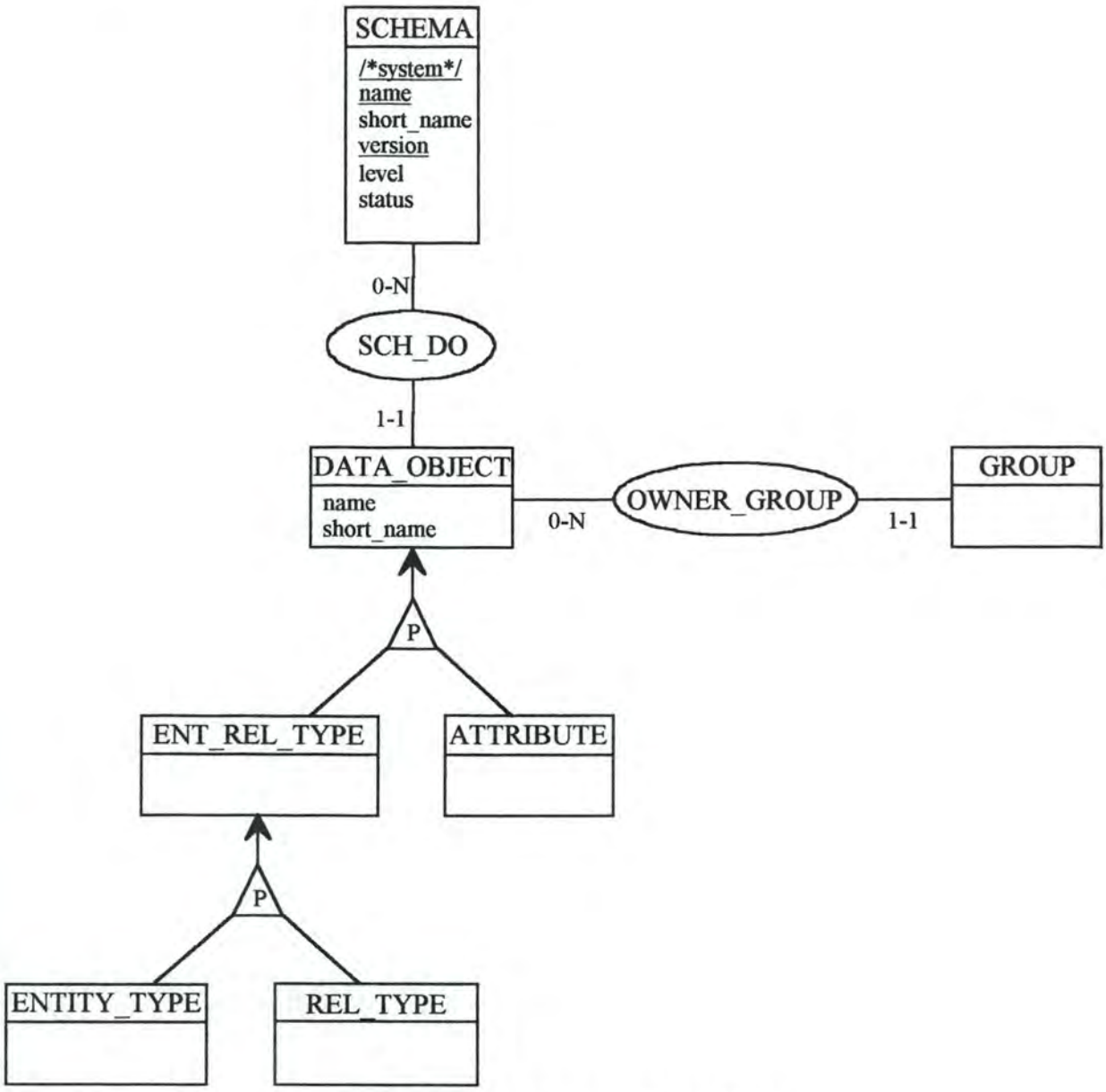


Figure 2.12 : Strucutre de base d'un schéma

L'élément de base d'un schéma est la donnée (**DATA_OBJECT**). Tout **DATA_OBJECT** appartient à un schéma (via **SCH_DO**). Le **DATA_OBJECT** est en fait la généralisation des notions de type d'entité (**ENTITY_TYPE**), type d'association (**REL_TYPE**) et attribut (**ATTRIBUTE**). Ces trois types d'objets spécifiques sont donc regroupés en un objet générique, dont ils forment une partition. Tous les attributs et les associations de l'objet générique ont donc une signification pour les objets spécifiques. Il est à noter que les notions de type d'entité et de type d'association sont regroupées dans un type d'entité générique appelé **ENT_REL_TYPE**.

Un **DATA_OBJECT** est caractérisé par un nom (***name***) et un nom court (***short_name***), et il possède un certain nombre de groupes (**GROUP**) via le type d'association **OWNER_GROUP**.

3.1.3 Les types d'entités et les types d'associations

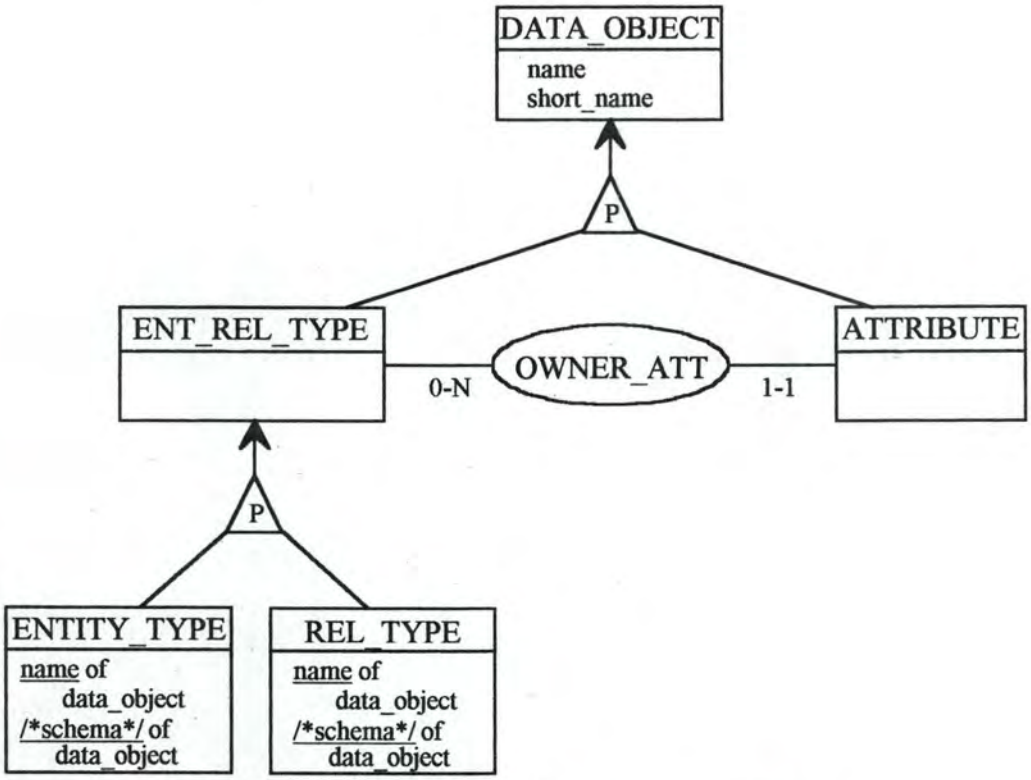


Figure 2.13 : Les types d'enittés et les types d'associations

Les types d'entités **ENTITY_TYPE** et **REL_TYPE** sont généralisés en un type d'entité générique **ENT_REL_TYPE**. Un **ENT_REL_TYPE** possède zéro ou plusieurs attributs (via **OWNER_ATT**).

Dans un **SCHEMA**, un type d'entité ou un type d'association est identifié par son nom (***name***) hérité du **DATA_OBJECT**.

3.1.4 Les liens entre les types d'entités et les types d'associations

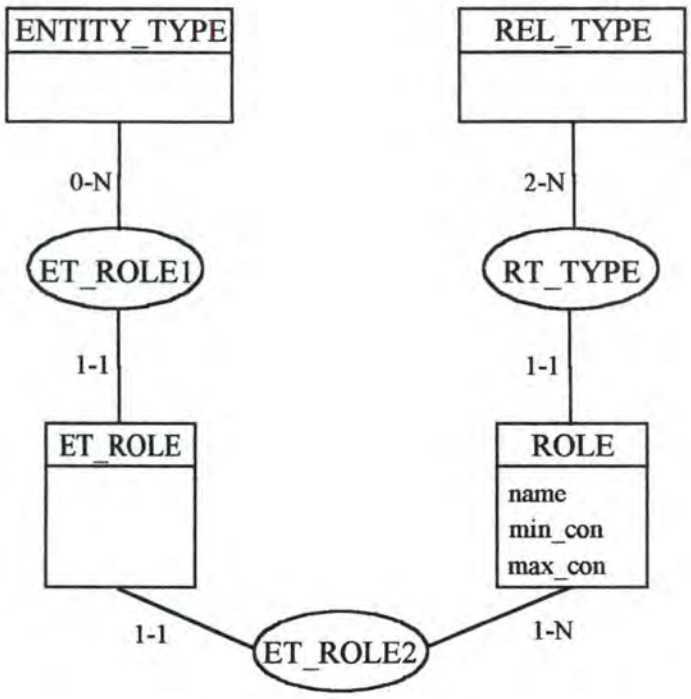


Figure 2.14 : Les rôles

Pour pouvoir apporter une solution à la représentation des rôles multi-domaines, la notion d'**ET_ROLE** a été introduite. En effet, le concept de rôle multi-domaines implique qu'un rôle peut être associé à plus d'un type d'entité, ce qui n'est pas le cas dans la représentation traditionnelle où un rôle ne peut être relié qu'à un et un seul type d'entité, et un et un seul type d'association. Graphiquement, l'**ET_ROLE** est une des branches de la "fourchette" représentant le rôle multi-domaines (figure 2.15).

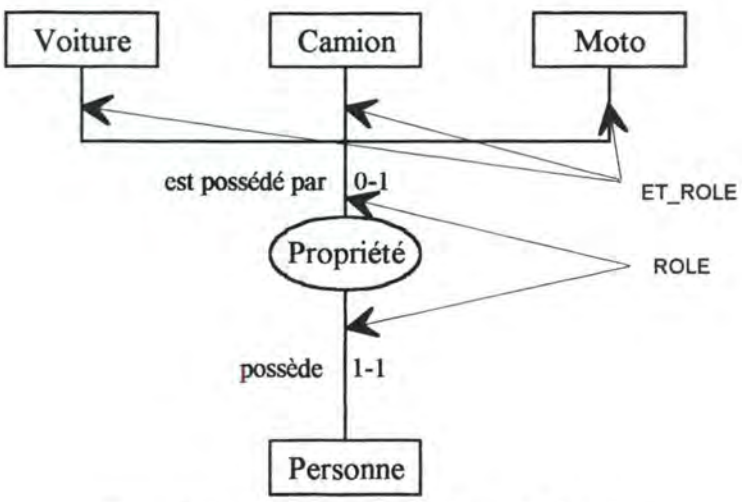


Figure 2.15 : Illustration des rôles multi-domaines

Un rôle est caractérisé par son nom (***name***), et par ses cardinalités minimum et maximum (***min_con*** et ***max_con***).

3.1.5 Les attributs

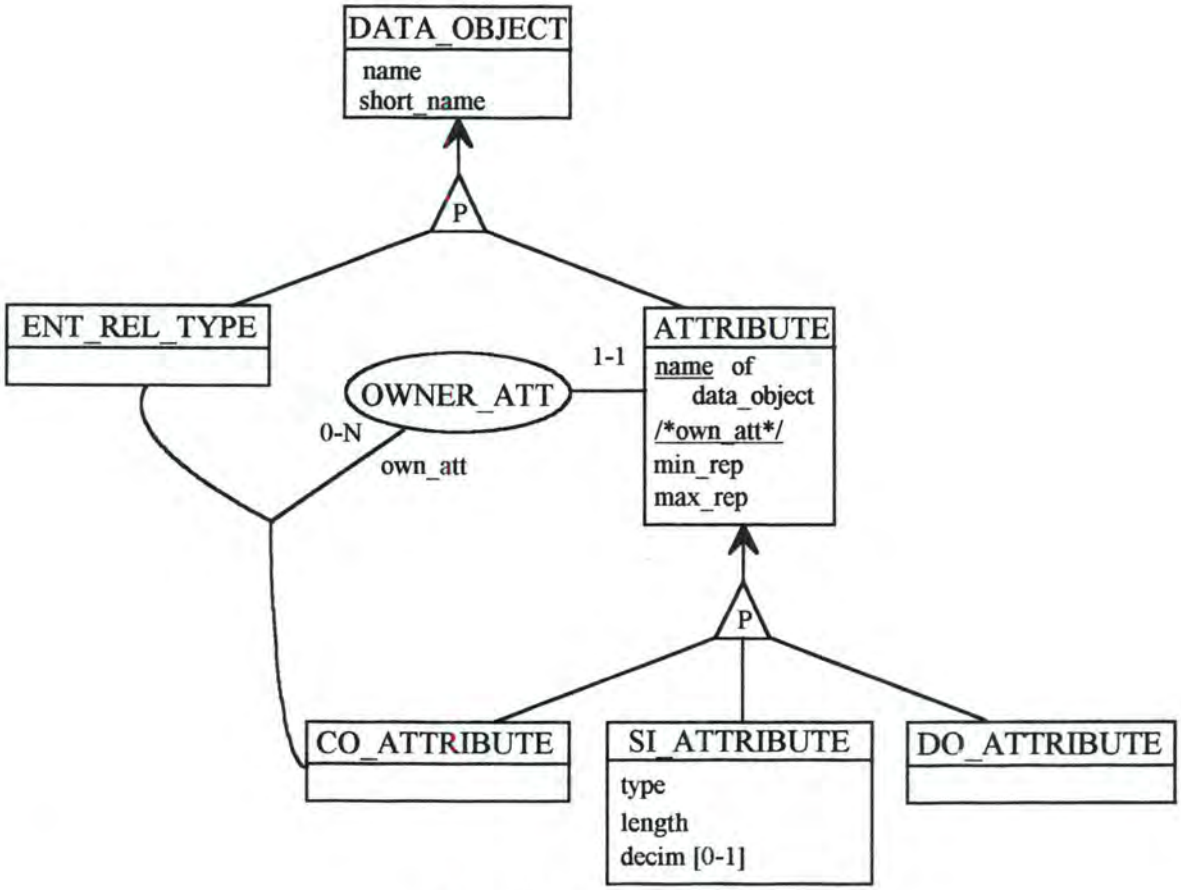


Figure 2.16 : Les attributs

Toute propriété caractérisant un type d'entité ou un type d'association est représentée par un attribut (**ATTRIBUTE**) qui lui est associé via **OWNER_ATT**.

Il existe trois types d'attributs : les attributs décomposables (**CO_ATTRIBUTE**), les attributs élémentaires (**SI_ATTRIBUTE**) et les attributs associés à un domaine particulier (**DO_ATTRIBUTE**). Ces trois objets sont la spécialisation de l'objet générique **ATTRIBUTE** dont ils forment une partition.

Un attribut est identifié par son nom (***name***) et par le type d'entité, le type d'association ou l'attribut auxquel il appartient. Il est caractérisé par ses cardinalités minimum et maximum (***min_rep*** et ***max_rep***).

Un attribut décomposable possède un ou plusieurs attributs (via **OWNER_ATT**). Il est à noter qu'il faut ajouter une contraintes à cette représentation : un attribut décomposable ne peut faire partie de ses composants.

Un attribut simple est caractérisé par un type (***type***) qui peut être *alphanumérique*, *numérique*, *date* ou *booléen*, par une longueur (***length***) et éventuellement (s'il est *numérique*) par un nombre de décimales (***decim***).

Un attribut de type domaine est associé à un domaine (**DOMAIN**) via **ATT_DOM** (voir §3.1.7).

3.1.6 Les groupes

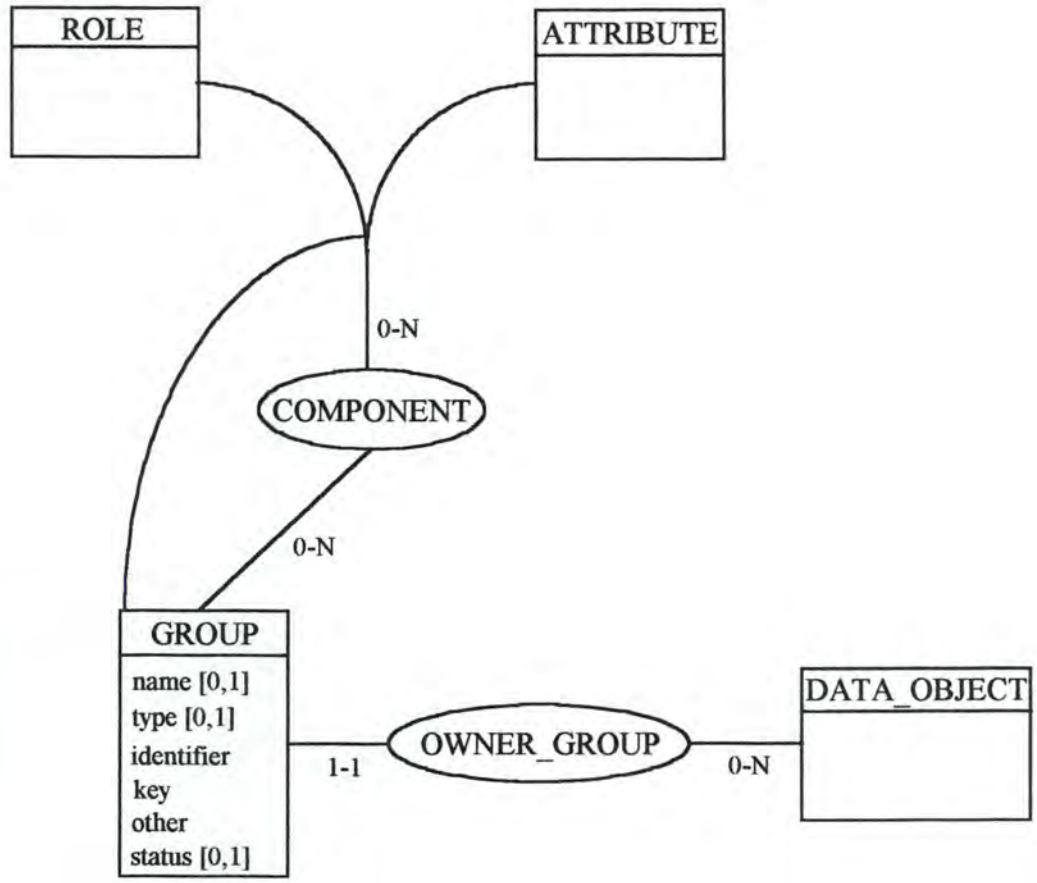


Figure 2.17 : Les groupes

Un groupe (**GROUP**) est un ensemble d'attributs (**ATTRIBUTE**) et/ou de rôles (**ROLE**) et/ou de groupes (**GROUP**). Chacun de ces attributs, rôles ou groupes peuvent appartenir à plusieurs groupes. Cette notion est modélisé par le type d'association **COMPONENT** qui relie un groupe à ses composants.

Un groupe est associé à un **DATA_OBJECT** via le type d'association **OWNER_GROUP**. Il est caractérisé par un nom (***name***) et un type (***type***). Le type indique s'il s'agit d'une composition ("C") d'attributs et/ou de rôles et/ou de groupes, c'est-à-dire d'un groupe appartenant à un autre groupe ou bien s'il s'agit d'une association ("A"), c'est-à-dire un groupe ayant une fonction dans le data_object auquel il est associé.

Les attributs ***identifier***, ***key*** et ***other*** sont des booléens qui indiquent la fonction (identifiant, clé d'accès ou autre) jouée par le groupe au sein du data_object. Le status (***status***) est également un booléen qui prend la valeur VRAIE si le groupe est identifiant primaire.

3.1.7 Les domaines

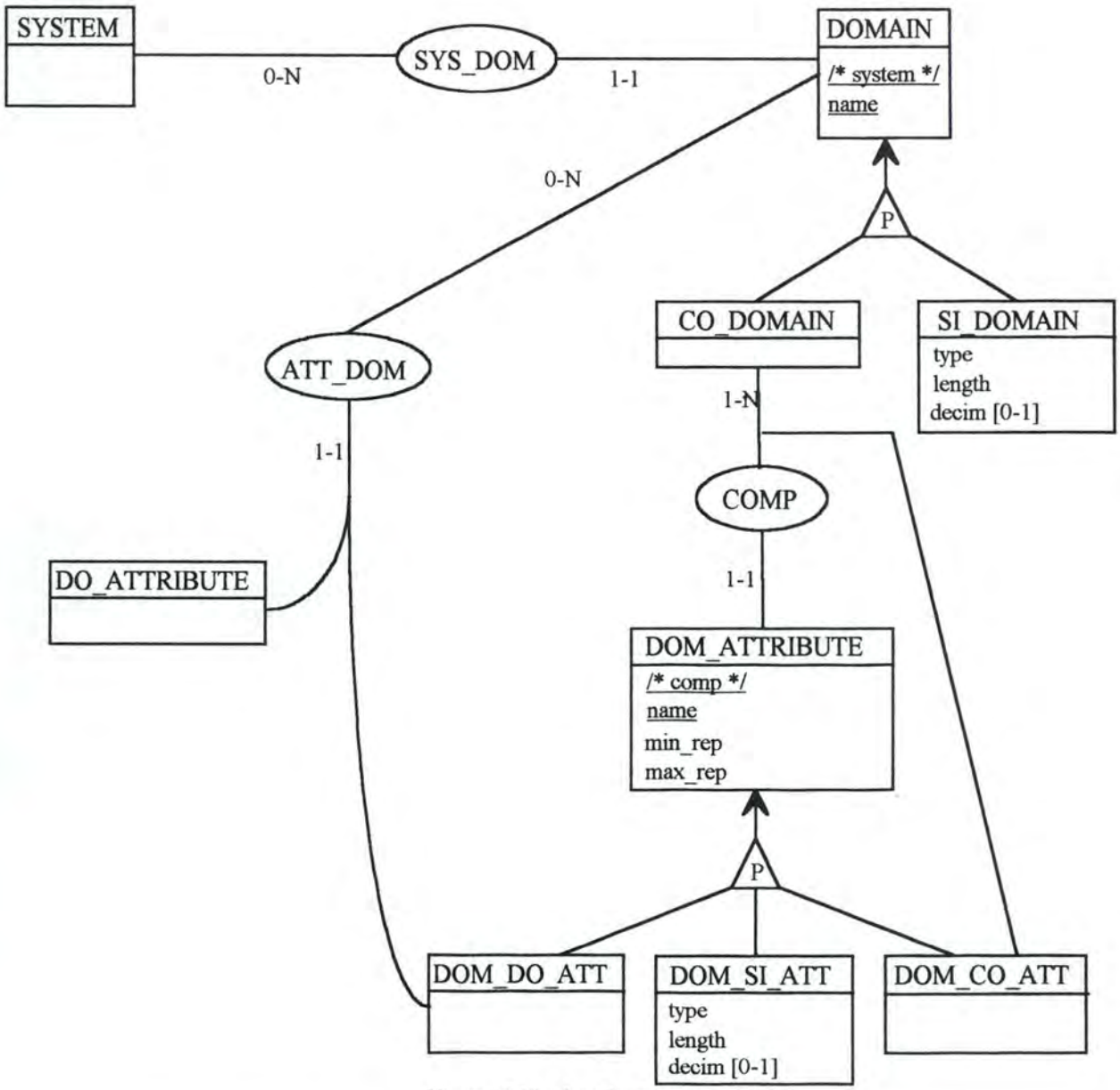


Figure 2.18 : Les domaines

Un domaine (**DOMAIN**) est attaché au système (**SYSTEM**) via **SYS-DOM**. Un attribut de type domaine (**DO_ATTRIBUTE**) est associé à un domaine via **ATT_DOM**.

Un domaine est spécialisé en deux sous-types : les domaines décomposables (**CO_DOMAIN**) et les domaines élémentaires (**SI_DOMAIN**). Les domaines élémentaires ont un type (***type***) qui peut être *alphanumérique*, *numérique*, *date* ou *booléen*, une longueur (***length***) et éventuellement un nombre de décimales (***decim***). Les domaines décomposables sont composés d'un ou plusieurs attributs domaines (**DOM_ATTRIBUTE**).

La structure d'un attribut domaine est fort semblable à celle d'un attribut. Il est caractérisé par un nom (***name***) qui l'identifie au sein d'un domaine décomposable (**CO_DOMAIN**), et par ses cardinalités minimum et maximum (***min_rep*** et ***max_rep***).

Il existe trois types d'attributs domaines : les attributs domaines décomposables (**DOM_CO_ATT**), les attributs domaines élémentaires (**DOM_SI_ATT**) et les attributs domaines de type domaine (**DOM_DO_ATT**). Ces trois objets sont la spécialisation de l'objet générique **DOM_ATTRIBUTE** dont ils forment une partition.

Un attribut domaine décomposable possède un ou plusieurs attributs domaines (**DOM_ATTRIBUTE**) via le type d'association **COMP**. Un attribut domaine ne peut faire partie de ses composants.

Un attribut domaine élémentaire est identique à un domaine élémentaire, il possède un type, une longueur et un nombre de décimale.

Un attribut domaine de type domaine est associé à un domaine (**DOMAIN**) via **ATT_DOM**.

3.1.8 Les valeurs

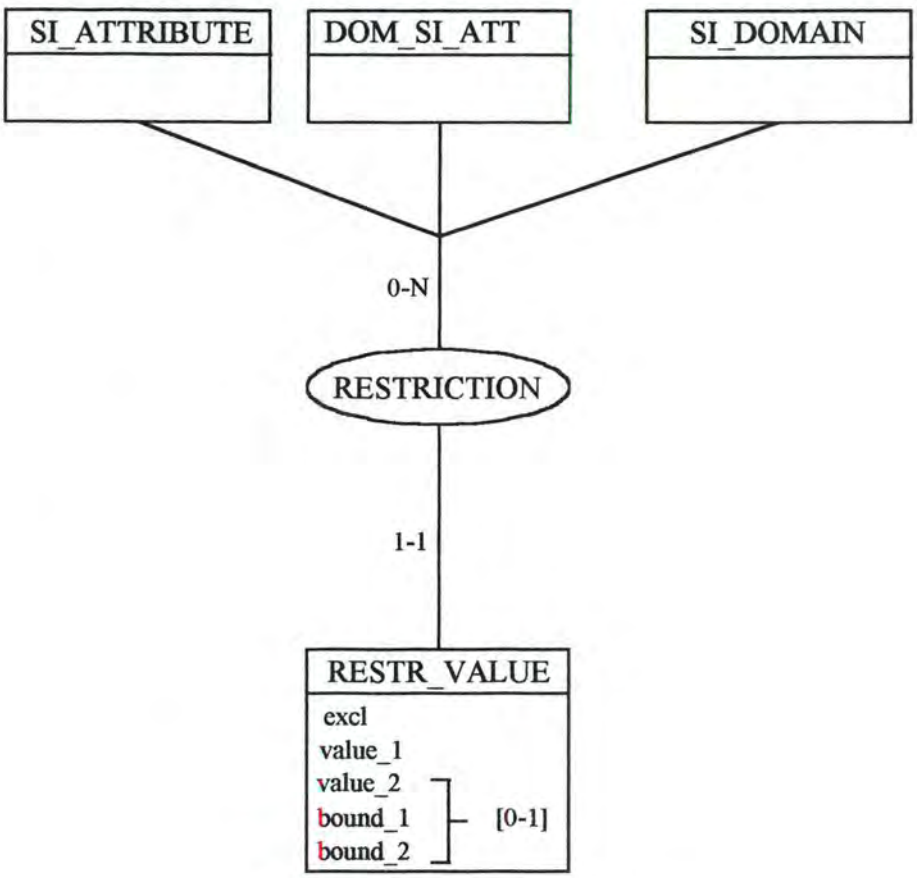


Figure 2.19 : Les valeurs restrictives

Une valeur restrictive (**RESTR_VALUE**) peut être associée à un attribut élémentaire (**SI_ATTRIBUTE**), un domaine élémentaire (**SI_DOMAIN**) ou un domaine attribut élémentaire (**DOM_SI_ATT**) via le type d'association (**RESTRICTION**). A chaque attribut ou domaine on peut associer un ensemble de valeurs (***value_1***) autorisées ou interdites selon la valeur de l'attribut ***excl***. On peut également associer un intervalle de valeurs autorisées ou interdites grâce aux

attributs ***value_1*** et ***value_2*** qui représentent les bornes de l'intervalle. Les attributs ***bound_1*** et ***bound_2*** représentent alors les types de bornes (ouvertes ou fermées) de ***value_1*** et ***value_2***.

3.1.9 Les contraintes

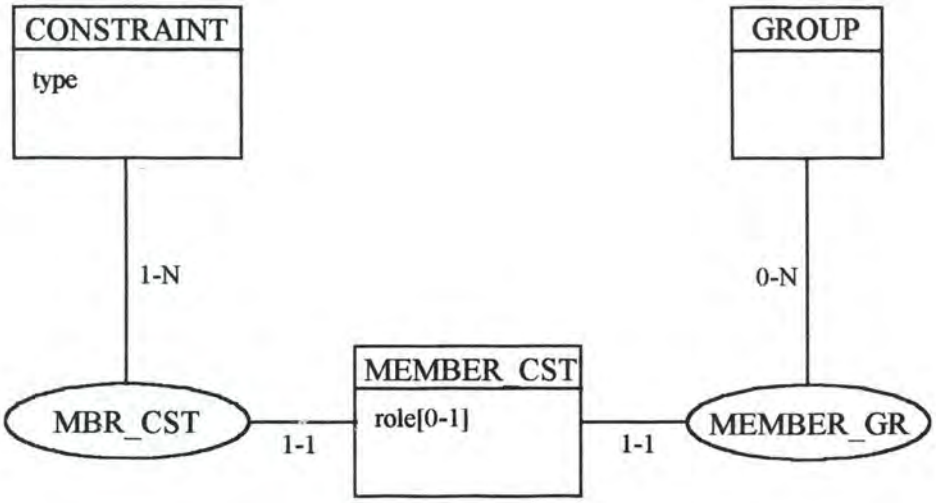


Figure 2.20 : Les contraintes

Seules les contraintes sur les groupes sont représentées. Un groupe (**GROUP**) peut être soumis à zéro ou plusieurs contraintes (**CONSTRAIN**). Le type d'entité **MEMBER_CST** exprime le lien entre le groupe et la contrainte.Un groupe peut avoir un rôle particulier dans une contrainte. Il peut être par exemple le groupe *origine* ou *cible* d'une contrainte d'inclusion ou bien le groupe *référencé* ou *référençant* d'une contrainte référentielle. Cette notion est exprimée grâce à l'attribut ***role*** du type d'entité **MEMBER_CST**.

Une contrainte est caractérisée par un type (***type***) qui permet d'indiquer s'il s'agit d'une inclusion ("I") ou d'une égalité ("=").

3.1.10 La Généralisation/Spécialisation

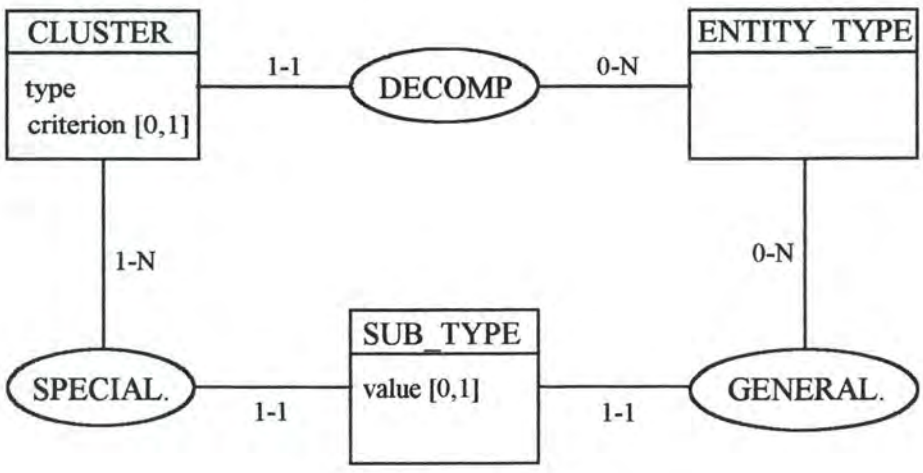


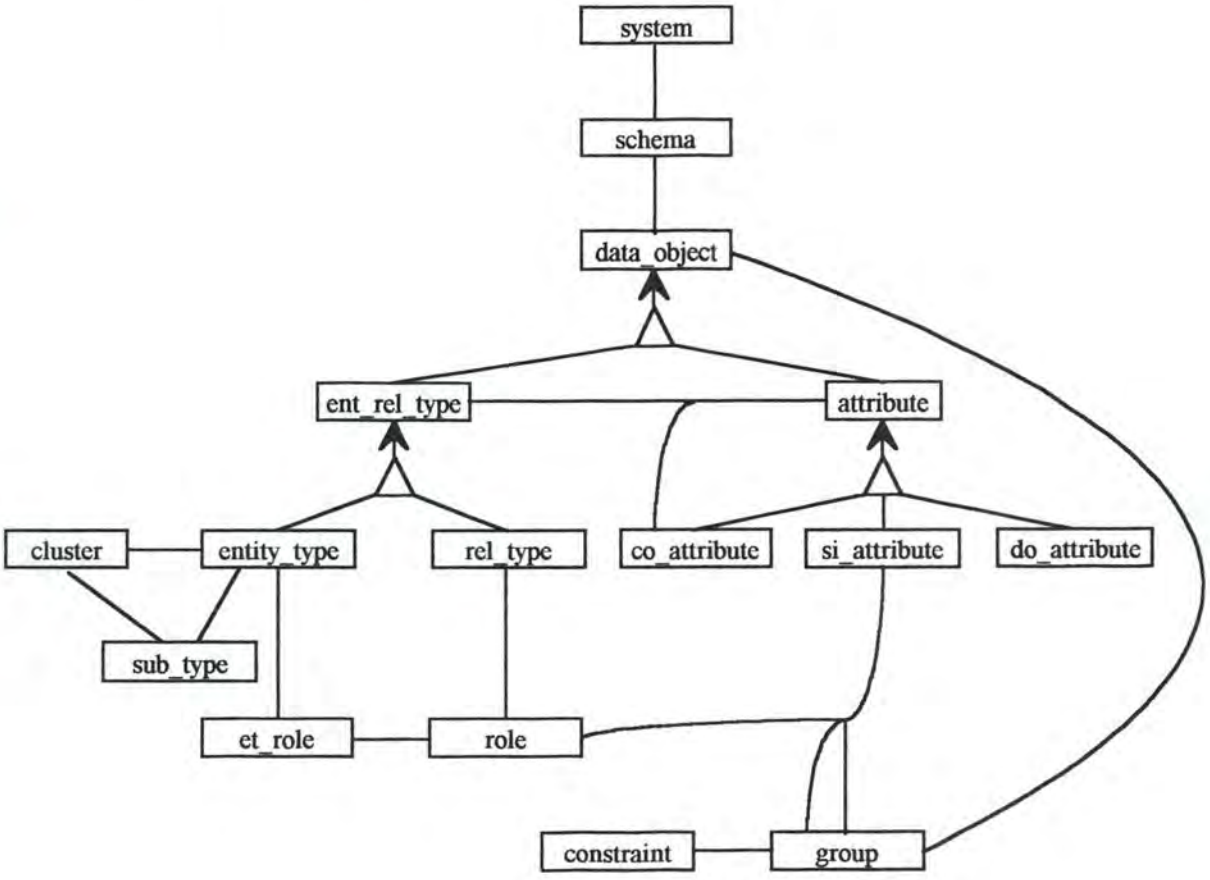
Figure 2.21 : Les clusters

Un type d'entité (**ENTITY_TYPE**) peut faire l'objet de zéro ou plusieurs décompositions via le type d'association (**DECOMP**). Un cluster (**CLUSTER**) permet de représenter un groupe de types d'entités spécifique, l'attribut ***type*** indique s'il s'agit d'une disjonction, d'une couverture ou d'une partition, et l'attribut ***criterion*** indique le critère de spécialisation. Par exemple le critère de spécialisation dans le cas d'un type d'entité PERSONNE spécialisé en HOMME et FEMME est le sexe.

Un cluster (**CLUSTER**) est relié à un ou plusieurs types d'entités (**ENTITY_TYPE**) via le type d'association **SUB_TYPE**. Le type d'association **SUB_TYPE** est caractérise par une valeur (***value***) qui indique l'instance du critère de cluster auquel on a affaire. Par exemple, la valeur du critère est soit "masculin", soit "féminin".

3.1.11 Schéma de synthèse

On reprend les types d'entités avec leur nom, ainsi qu'une représentation linéaire des types d'associations. Un cadre grisé représente un type d'entité déjà décrit.



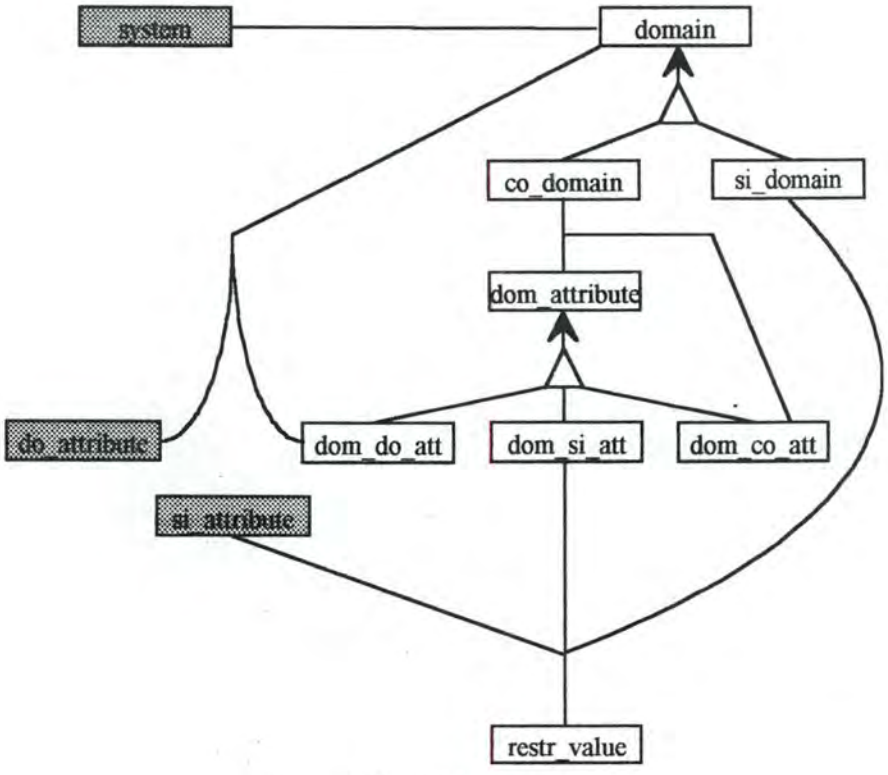


Figure 2.22 : Schéma de synthèse

3.2 Techniques d'implémentation

Le schéma Entité/Association de la base des spécifications présenté dans la section précédente a été conçu dans une perspective d'implémentation orientée objet. Ce Schéma peut donc facilement être traduit en un schéma orienté objet moyennant quelques techniques particulières.

3.2.1 Correspondance entre les concepts E/A et Orientés Objets

Les concepts utilisés sont pratiquement les mêmes, seule la terminologie change.

Chaque type d'entité du schéma est une classe d'objet et ses attributs sont des propriétés de la classe d'objets.

Les relations d'héritage de la conception orientée objet représentent le même concept que les relations de Généralisation/Spécialisation. Un cluster dans le schéma E/A devient donc une relation d'héritage entre les classes d'objets qu'il relie.

Tous les types d'associations complexes (association N-N) du schéma sont transformés en type d'entité. Ils représentent donc également des classes d'objets.

Tableau récapitulatif des correspondances :

Schéma E/A		Schéma OO
Type d'entité	$\Rightarrow$	Classe d'objets
Type d'association complexe	$\Rightarrow$	Classe d'objets
Attribut	$\Rightarrow$	Attribut de classe
Cluster	$\Rightarrow$	Relation d'héritage

3.2.2 Les types d'associations 1-N

Les types d'associations sont représentés par un mécanisme de référence du type de celui utilisé dans les bases de données de type réseau.

Chaque objet origine d'une association (côté N du type d'association) contient une référence vers le premier objet cible de l'association et une autre vers le dernier.

Chaque objet cible d'une association (côté 1 du type d'association) contient une référence vers l'objet cible suivant de l'association, une référence vers l'objet cible précédent et une référence vers l'objet origine.

3.2.2 Les propriétés et les classes d'objets supplémentaires

Tous les objets d'une classe terminale sont chaînés entre eux. Cela signifie que chaque objet contient une référence vers l'objet suivant de sa classe.

Chaque objet du système contient un identifiant interne qui permet de l'identifier parmis tous les objets de la base de données.

Aux classes d'objets définies dans le schéma conceptuel, on a ajouté une classe d'objet générique qui constitue le sommet de la hiérarchie des classes d'objets. Cette classe appelée **GENERIC_OBJECT** a donc permis de regrouper des propriétés communes à toutes les classes du schéma telles que l'identifiant interne et la référence vers l'objet suivant de la classe.

3.2.3 La hiérarchie des classes d'objets

La figure 2.23 représente le schéma complet de la hiérarchie des classes d'objets de la base des spécifications

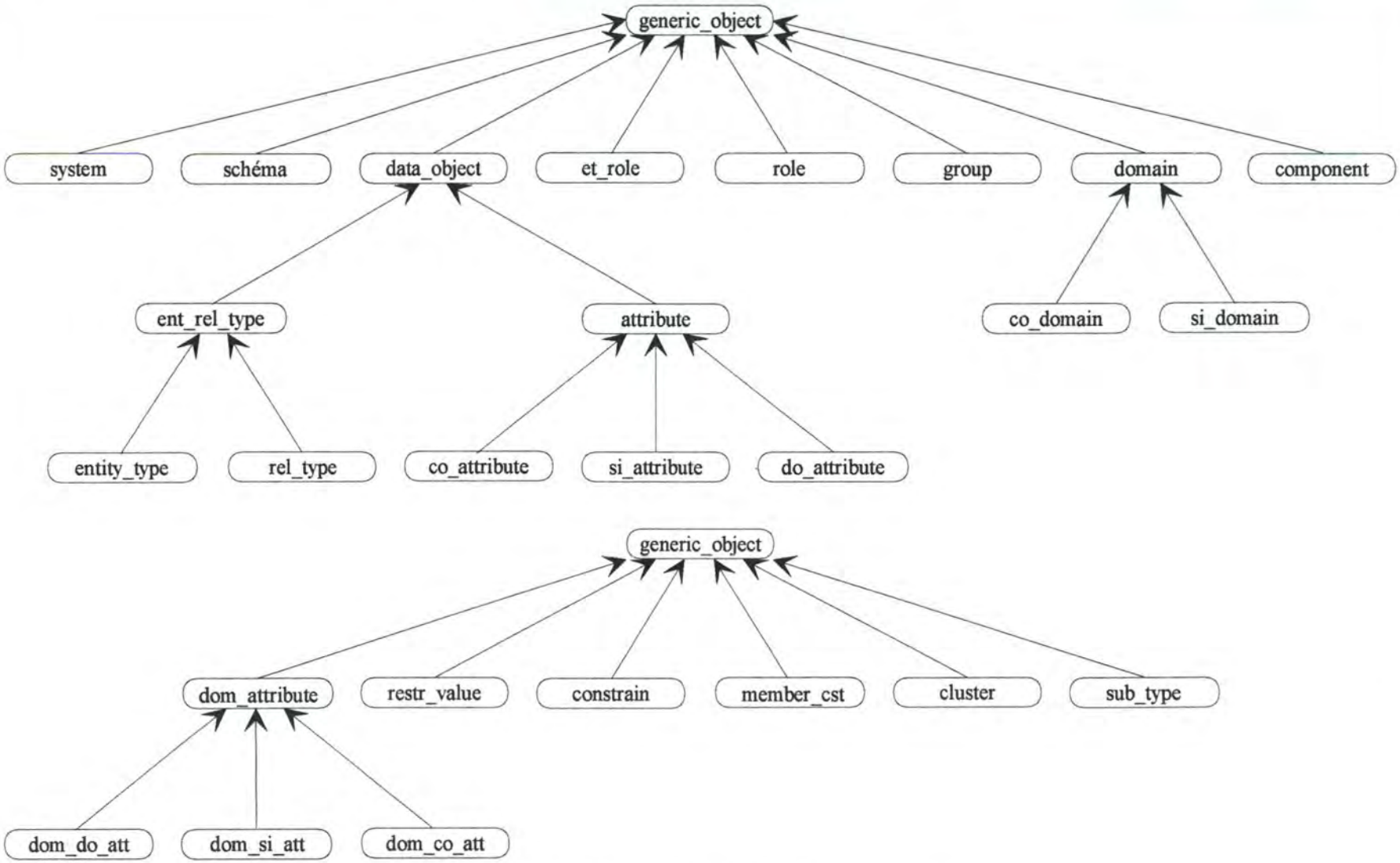


Figure 2.23 : Schéma de la hiérarchie des classes d'objets

3.2.4 Les interfaces des classes d'objets

A chaque classe d'objets du schéma correspond un ensemble de méthodes qui constituent l'interface de l'objet.

Ces méthodes sont les primitive d'accès classiques, rencontrées lors de l'élaboration de tout module d'accès à une base de données :

- Les primitives de création et de suppression d'objets (constructeurs et destructeurs)
- Les primitives de consultation et de mise à jour des attributs des classes d'objets. Ces attributs étant décaré comme privé à une classe, on ne peut y accéder que par ces méthodes.
- Les primitives d'accès au premier objet cible ou à l'objet cible suivant d'un type d'association. Par exemple, accès au premier attribut d'un type d'entité.
- Les primitives d'ajout et de suppression d'objets cibles d'un type d'association.

A ces fonctions, on peut ajouter des fonctions particulières à des classes d'objets telles que des fonctions de transformation d'objet ou des primitives qui renvoie des propriétés particulières d'un objet (par exemple : le degré d'un type d'association).

Un descritption complète de ces primitives peut être consultée dans [DEMA-92].

Chapitre 3 : Les outils de transformations

Introduction

Ce chapitre présente les outils de transformations développés.

Lors du développement de la base des spécifications présentée au chapitre 2, seul un sous-schéma du schéma conceptuel a été implémenté. Ce sous-schéma reprend les notions de systèmes, de schémas, de types d'entités, de types d'associations et rôles, de groupes et de contraintes référentielles. Pour la spécification et l'implémentation des outils de transformations et de génération présentés dans ce chapitre et dans le chapitre suivant, seules ces notions ont été prises en compte.

La première section de ce chapitre, **Définition et objectifs des transformations**, introduit la notion de transformation.

La deuxiéme section, **Les types de transformations**, présente les trois types de transformations proposés dans l'atelier logiciel.

La troisième section, **Les transformations élémentaires**, donne une spécification complète de quatres transformations et leurs transformations inverses.

La dernière section, **Transformation vers le modèle relationnel**, aborde le problème de la transformation d'un schéma vers un modèle de donné particulier.

1. Définition et objectifs des transformations

1.1 Notion de transformation

Une transformation est une opération qui modifie un schéma tout en préservant un certain nombre d'aspects conceptuels et techniques. Cette préservation est garantie par l'existence d'une opération inverse autorisant le retour au schéma initial. D'une manière générale, une transformation permet de supprimer d'un schéma une construction indésirable, soit parce qu'il s'agit d'une anomalie de représentation (par exemple : transformation des attributs répétitifs décomposables en types d'entités), soit parce la construction est non conforme à un SGBD particulier (par exemple : transformation des types d'associations en attributs de référence).

1.2 Définition

On peut définir une transformation de la façon suivante [HCDM-91] :

Une transformation T est une opération qui remplace une construction C d'un schema S par une autre construction C'. C' est la cible de C via T, et est notée **C' = T(C)**.

Une transformation T est définie par,

(1) une précondition P que toute construction C doit satisfaire pour pouvoir être transformée par T,

(2) une postcondition Q que T(C) satisfait.

T peut également s'écrire T = <P,Q>. P et Q sont des prédicats qui identifient les composants et les propriétés de C et T(C), et plus particulièrement :

- les composants de C qui sont préservés dans T(C),
- les composants de C qui n'apparaissent plus dans T(C),
- les composants de T(C) qui n'existaient pas dans C.

Une transformation T1 = <P,Q> est réversible si et seulement si il existe une transformation T2 telle que, pour tout schema S, $P(S) \Rightarrow T2(T1(S)) = S$

T2 est l'inverse de T1, et réciproquement. On a la propriété suivante : T2 = <Q,P>.

2. Les types de transformations

Dans un atelier logiciel, il est intéressant de proposer à l'utilisateur plusieurs types de transformations qui vont lui permettre d'orienter son processus de conception.

2.1 Les transformations élémentaires

Les transformations élémentaires consistent à appliquer une transformation à un objet du schéma.

Avec ce type de transformation, l'utilisateur garde le contrôle du processus de transformation de son schéma. En effet, à une situation correspond souvent plusieurs solutions et l'utilisateur peut faire le choix de la solution qui lui paraît la plus adaptée. Par exemple, un attribut répétitif peut être transformé de plusieurs façons différentes.

Les transformations élémentaires développées sont les suivantes : transformation d'un type d'entité en type d'association, transformation d'un attribut en type d'entité, transformation d'un type d'association en groupe de référence et désagrégation d'un attribut décomposable, avec pour chacune d'entre elles leurs transformations inverses. On peut trouver une description complète de ces transformations dans la troisième section de ce chapitre.

2.2 Les transformations globales

Les transformations globales consistent à appliquer une transformation à tous les objets auxquels cette transformation peut être appliquée (tous les objets qui satisfont la précondition de la transformation). Ce type de transformation permet d'éliminer en une fois des structures non compatibles avec un modèle particulier. Par exemple, désagrégation de tous les attributs décomposables du schéma.

Les transformations globales développées sont : désagrégation des attributs décomposables, transformations des attributs répétitifs en types d'entités, transformations des types d'associations en types d'entités et transformation des types d'associations en attributs de référence.

2.3 Les transformations selon un modèle

Les transformations selon un modèle consistent à transformer toutes les constructions d'un schéma qui ne satisfont pas les règles d'un modèle de données particulier. Ce type de transformation ne demande aucun contrôle de l'utilisateur, le schéma obtenu est correct et conforme au modèle choisi.

La transformation selon un modèle développée est la transformation vers le modèle relationnel, elle est présentée dans la dernière section de ce chapitre.

3. Les transformations élémentaires

Cette section décrit une série de transformations élémentaires. Ces transformations sont en fait 4 transformations usuelles et leurs transformations inverses :

- Transformation d'un type d'association en type d'entité, et transformation d'un type d'entité en type d'association;
- Transformation d'un attribut en type d'entité, et transformation d'un type d'entité en attribut;
- Transformation d'un type d'association en groupe de référence, et transformation d'un groupe de référence en type d'association;
- Désagrégation d'un attribut décomposable, et agrégation d'un ensemble d'attributs;

Pour chacune de ces transformations, nous donnons une description succincte et un exemple explicatif. Ensuite il y a une définition plus détaillée à la fois sous forme graphique et textuelle. La partie textuelle est composée d'une précondition et d'un ensemble d'actions. La précondition décrit un ensemble de règles que l'objet doit respecter pour pouvoir être transformé et l'ensemble d'actions décrit sous forme de prédicats le processus de transformation. Ce formalisme de description est à distinguer du formalisme présenté dans la première section de ce chapitre où les préconditions (P) et postconditions (Q) sont constituées d'un ensemble d'objets et de propriétés du schéma avant et après la transformation.

3.1 Transformation d'un type d'association en type d'entité

3.1.1 Description

Un type d'association est transformé en type d'entité et chacun de ses rôles en type d'association.

Cette transformation est utilisée pour obtenir des structures de données sans types d'associations N-N et des types d'associations sans attributs. Elle est très souvent utilisée car très peu de SGBD accepte ce genre de structure.

3.1.2 Exemple

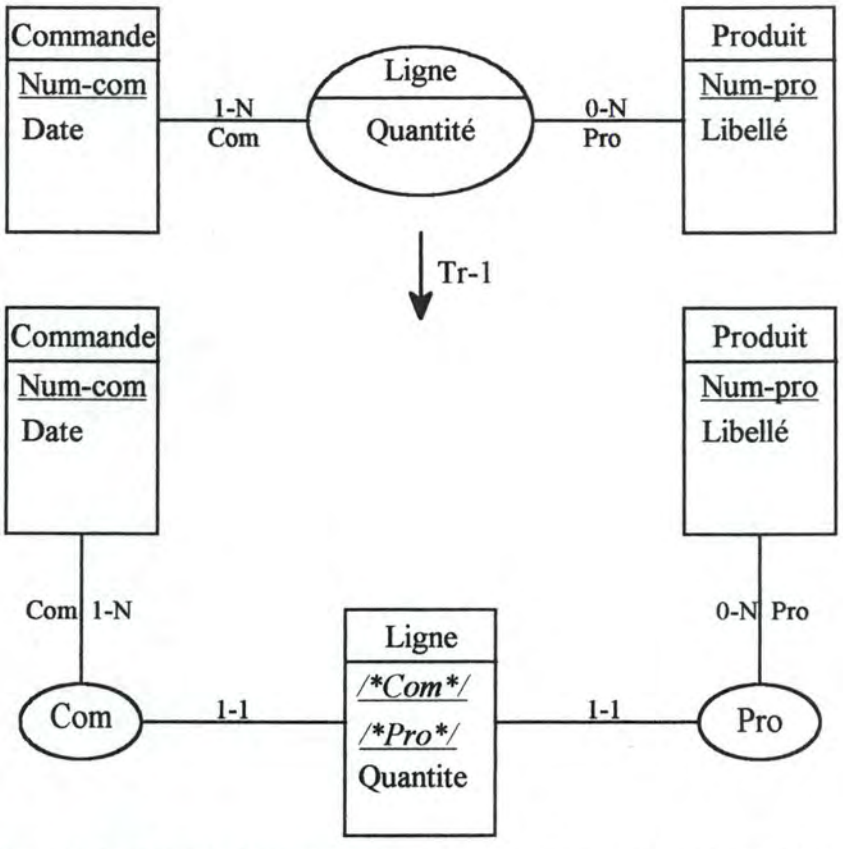


Figure 3.1 : Exemple de transformation d'un type d'association en type d'entité

3.1.3 Définition

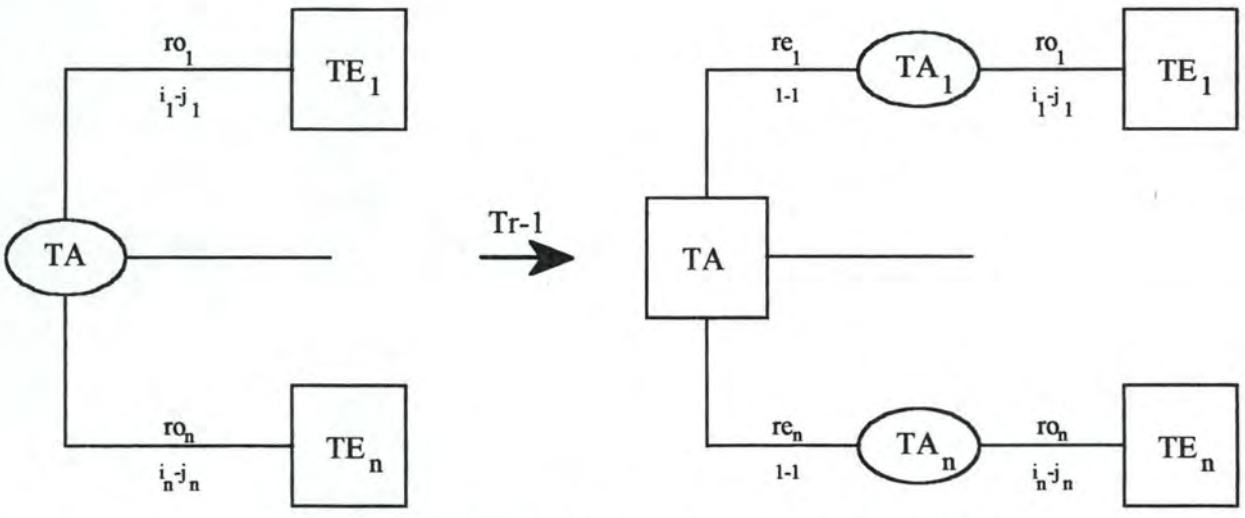


Figure 3.2 : Transformation d'un type d'association en type d'entité

Préconditions :

- Les rôles du type d'association ne peuvent appartenir qu'à des groupes du type d'association.

Actions :

1. Remplacer le type d'association TA par un type d'entité avec le même nom, les mêmes attributs et les mêmes groupes.
2. Chaque rôle ro_i (qui peut être multi-domaine) de TA est remplacé par un type d'association binaire avec :
   - le nom de ro_i s'il en a un, un nom composé de la concaténation des noms courts de TA et TE_i sinon,
   - un rôle 1-1 qui le relie à TA,
   - un role ro_i qui le relie à TE_i et qui a la même cardinalité que ro_i, les mêmes participations à des groupes que ro_i, les mêmes types d'entité que ro_i (cas des rôles multi-domaines).
3. Si le type d'association n'avait pas de groupe identifiant et si aucun rôle n'a de cardinalité maximum à 1 alors tous les ro_i font partie du groupe identifiant,

3.2 Transformation d'un type d'entité en type d'association

3.2.1 Description

Un type d'entité est transformé en type d'association et toutes les relations sont transformées en rôles.

Cette transformation est la transformation inverse de la transformation présentée au §3.1.

3.2.2 Exemple

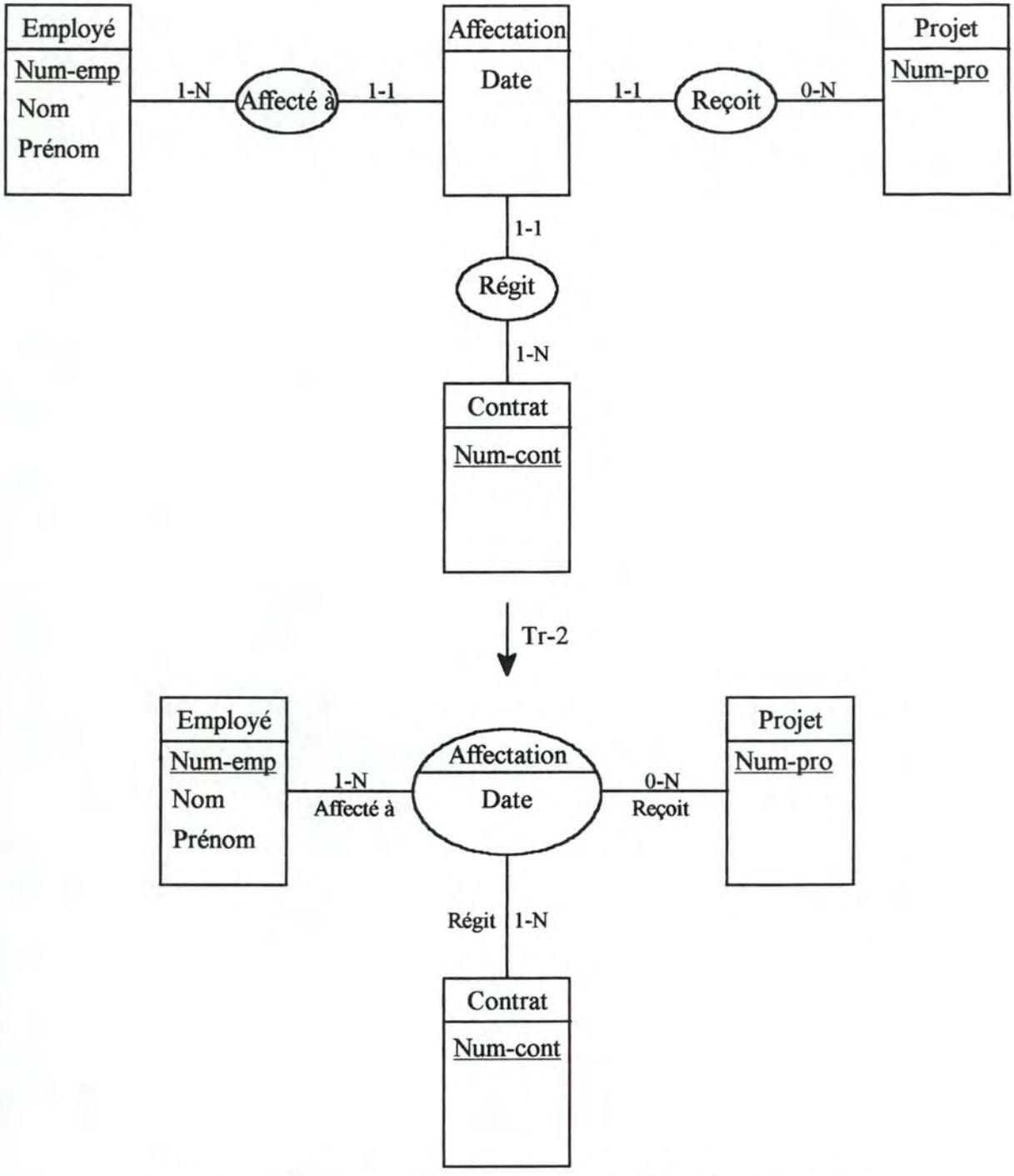


Figure 3.3 : Exemple de transformation d'un type d'entité en type d'association

3.2.3 Définition

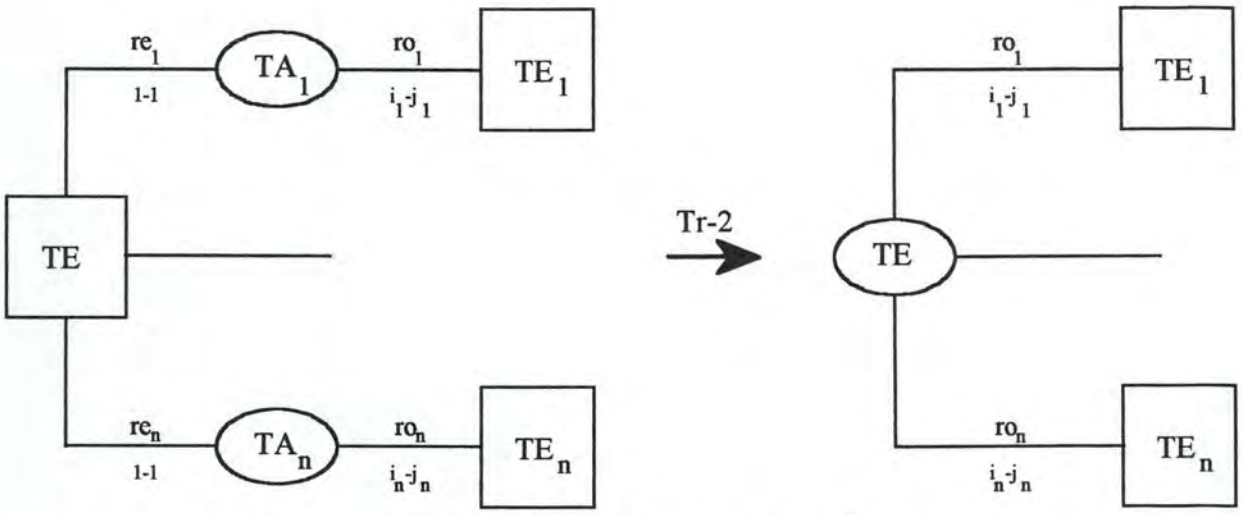


Figure 3.4 : Transformation d'un type d'entité en type d'association

Préconditions :

- TE doit jouer un rôle dans au moins deux types d'associations,
- Tous les rôles joués par TE doivent avoir une cardinalité 1-1 et être mono-domaines,
- Tous les types d'associations reliés à TE doivent être binaires, non cycliques et sans attributs,
- Les rôles joués par TE ne peuvent être composants d'aucun groupe.

Actions :

1. Remplacer le type d'entité TE par un type d'association avec le même nom, les mêmes attributs et les mêmes groupes.
2. Chaque type d'association TA_i est remplacé par un rôle avec :
   - le nom de TA_i,
   - les mêmes cardinalités que ro_i,
   - les mêmes participations à des groupes que ro_i,
   - les mêmes participations à des types d'entités que ro_i (cas des rôles multi-domaines).

3.3 Transformation d'un attribut en type d'entité

3.3.1 Description

Un attribut d'un type d'entité est transformé en type d'entité qui est relié à son entité origine par un type d'association. Il existe deux possibilités pour cette transformation : Soit chaque entité du nouveau type d'entité représente une valeur de l'attribut, soit chaque entité représente une instance particulière d'une valeur de l'attribut. On parle alors respectivement de transformation *orientée inventaire* (ou N-N) et de transformation *orientée valeur* (ou 1-N).

Cette transformation permet d'éliminer les attributs répétitifs et les attributs décomposables dans la perspective où le SGBD cible n'accepte pas ces structures.

3.3.2 Exemples

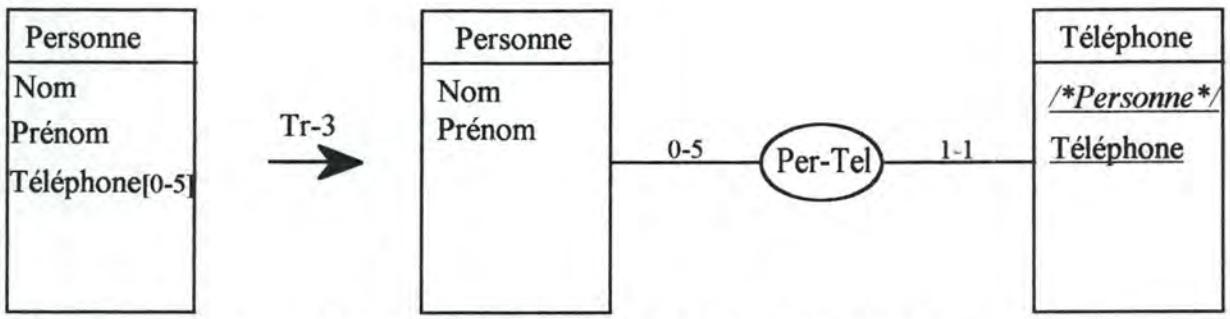


Figure 3.5 : Exemple de transformation d'un attribut en type d'entité (orientée valeur sur un attribut répétitif et élémentaire)

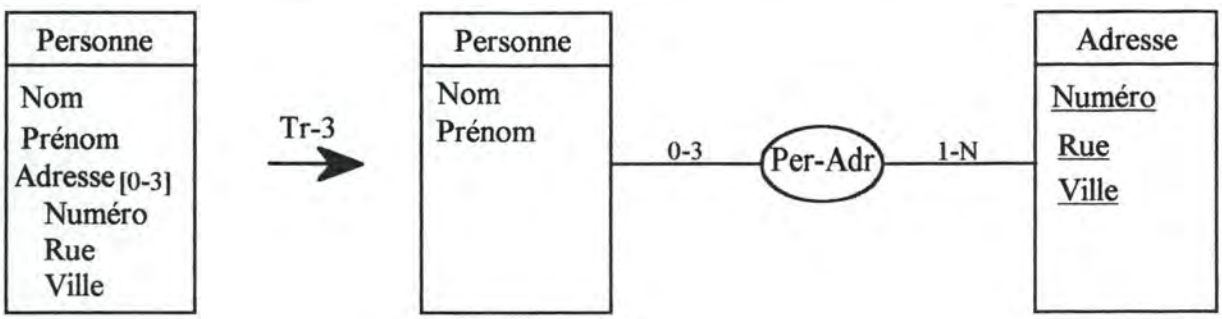


Figure 3.6 : Exemple de transformation d'un attribut en type d'entité (orientée inventaire sur un attribut répétitif et décomposable)

3.3.3 Définition

3.3.3.1 L'attribut est élémentaire

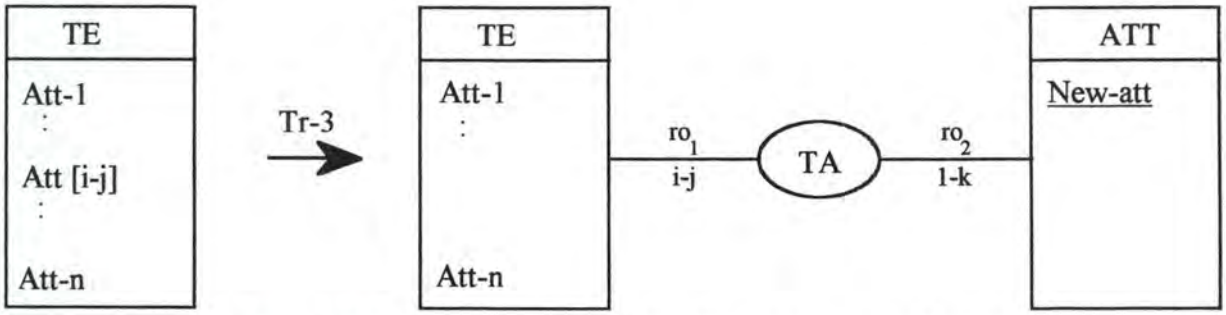


Figure 3.7 : Transformation d'un attribut élémentaire en type d'entité

A. Transformation orientée inventaire

Préconditions :

- Att est un attribut du premier niveau,
- Si Att est répétitif, alors Att ne peut faire partie d'aucun groupe multi-composants.

Actions :

1. Créer un type d'entité ATT, avec le même nom que Att.
2. Créer un attribut New-att dans le type d'entité ATT, sa cardinalité est égale à 1-1 et il est identifiant de ATT.
3. Créer un type d'association TA entre ATT et TE :
   - le nom de TA est la concaténation des noms courts de ATT et TE,
   - card (ro_1) = card (Att)[1],
   - min-card (ro_2) = 1[2],
   - Si Att identifie TE max-card (ro_2) = 1, sinon max-card (ro_2) = n,
   - Toutes les participations à des groupes multi-composants de Att dans TE sont remplacées par ro_2.
   - Toutes les participations à des groupes mono-composants non identifiants de Att dans TE sont remplacées par ro_2.
4. Supprimer Att.

[1] card(x) désigne la cardinalité de x, x étant un attribut ou un rôle
[2] min-card(x) et max-card(x) désignent la cardinalité minimum et maximum de x, x étant un attribut ou un rôle

B. Transformation orientée valeur

Préconditions :

- Att est un attribut du premier niveau,
- Att ne peut faire partie d'aucun groupe multi-composants,
- Att n'est pas identifiant.

Actions :

1. Créer un type d'entité ATT, avec le même nom que Att,
2. Créer un attribut New-att dans le type d'entité ATT, dont la cardinalité est 1-1.
3. Créer un type d'association TA entre ATT et TE :
   - le nom de TA est la concaténation des noms courts de ATT et TE.
   - card (ro_1) = card (Att),
   - min-card (ro_2) = 1,
   - max-card (ro_2) = 1,
   - toutes les participations à des groupes mono-composants non identifiants de Att dans TE sont remplacées par ro_2.
4. Créer un groupe identifiant du nouveau type d'entité ATT, composé de l'attribut Att et du rôle ro_1,
5. Supprimer Att.

3.3.3.2 L'attribut est décomposable

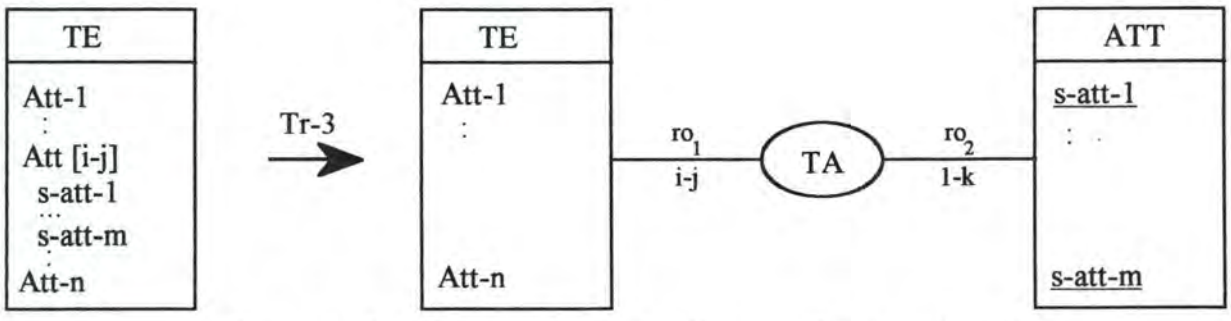


Figure 3.8 : Transformation d'un attribut décomposable en type d'entité

A. Transformation orientée inventaire

Préconditions :

- Att est un attribut du premier niveau,
- Si Att répétitif, alors Att ou un sous-ensemble de ses composants ne peut faire partie d'un groupe qui contiendrait d'autres composants.

Actions :

1. Créer un type d'entité ATT, avec le même nom que Att,
2. Transférer les attributs s-att-1,... s-att-m composant de Att dans le type d'entité ATT, ces attributs conservent leur répétitivité.
3. Créer un type d'association TA entre ATT et TE :
   - le nom de TA est la concaténation des noms courts de ATT et TE.
   - card (ro_1) = card (att),
   - min-card (ro_2) = 1,
   - Si Att ou un sous-ensemble de s-att-1,... s-att-m identifie TE,

     Alors max-card (ro_2) = 1,

     Sinon max-card (ro_2) = n,
   - Chaque groupe de TE composé uniquement d'un sous-ensemble des composants de Att est transféré dans le nouveau type d'entité ATT,
   - Toutes les participations à des groupes multi-composants de Att dans TE sont remplacées par ro_2,
   - Si un sous-ensemble de s-att-1,... s-att-m apparaît dans un groupe de TE avec d'autres composants, alors remplacer ces attributs par ro2.
4. Si le type d'entité ATT n'a pas encore de groupe identifiant alors créer un groupe identifiant de ATT et composé de s-att-1,... s-att-m.
5. Supprimer Att.

B. Transformation orientée valeur

Préconditions :

- Att est une attribut direct de TE,
- Att ou un sous-ensemble de ses composants ne peut faire partie d'un groupe qui contiendrait d'autres composants,
- Att ou un sous-ensemble de ses composants n'est pas identifiant de TE.

Actions :

1. Créer un type d'entité ATT, avec le même nom que Att,
2. Transférer les attributs s-att-1,... s-att-m composant de Att dans le type d'entité ATT, ces attributs conservent leur répétitivité.
3. Créer un type d'association TA entre ATT et TE :
   - le nom de TA est la concaténation des noms courts de ATT et TE,
   - card (ro_1) = card (Att),
   - min-card (ro_2) = 1,
   - max-card (ro_2) = 1,
   - Chaque groupe de TE composé uniquement d'un sous-ensemble des composants de Att est transféré dans le nouveau type d'entité ATT.
4. Si Le type d'entité ATT n'a pas encore de groupe identifiant :
   - Créer un groupe identifiant du nouveau type d'entité ATT, composé des attributs s-att-1,... s-att-m de ATT et du rôle ro_1,
5. Supprimer Att.

3.4 Transformation d'un type d'entité en attribut

3.4.1 Description

Un type d'entité qui est associé à un et un seul autre type d'entité est transformé en un attribut de ce type d'entité.

Cette transformation est la transformation inverse de la transformation présentée au §3.3.

3.4.2 Exemple

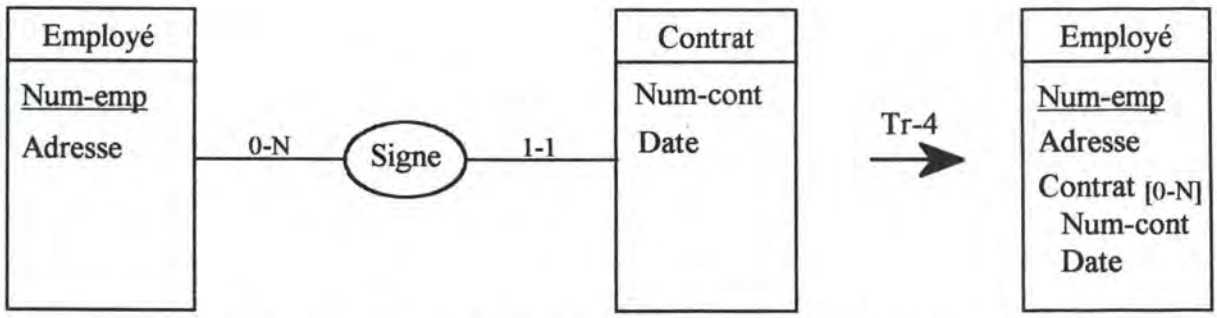
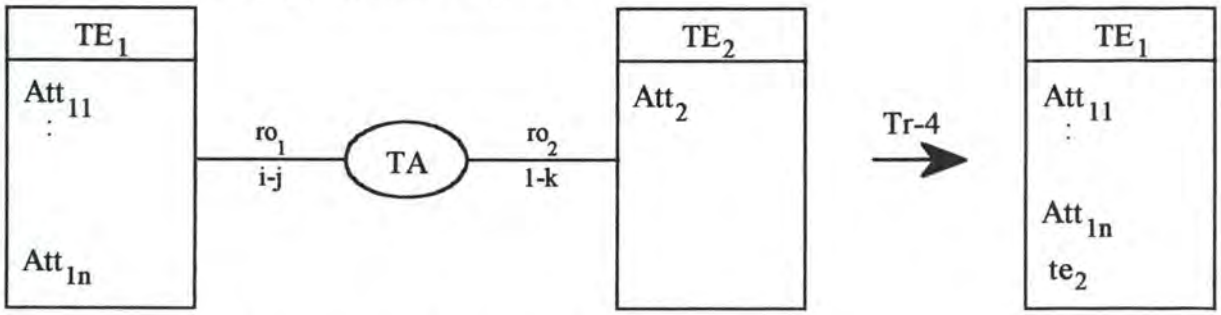


Figure 3.9 : Exemple de transformation d'un type d'entité en attribut

3.4.3 Définition

Cas 1 : Le type d'entité n'a qu'un attribut



Cas 2 : Le type d'entité a plusieurs attributs

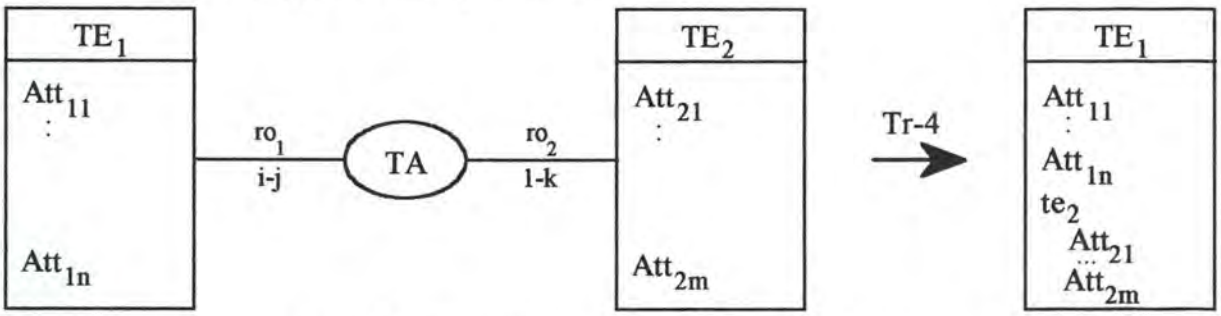


Figure 3.10 : Transformation d'un type d'entité en attribut

Préconditions :

- TE_1 et TE_2 sont reliés par un et un seul type d'association TA sans attribut, avec des rôles mono-domaines. La cardinalité minimale du role ro_2 joué par le type d'entité TE_2 dans la type d'association TA doit être 1.
- TE_2 a au moins un attribut.

- TE_2 a au moins un groupe identifiant. Tous les groupes de TE_2 sont composés uniquement d'attributs locaux et éventuellement de ro_1.
- Si TE_2 n'a qu'un seul attribut, soit la cardinalité du rôle ro_1 est 1-1, soit la cardinalité de l'attribut att_2 est 1-1, ou les 2.

Actions :

1. Créer un attribut te_2 dans TE_1, avec le même nom que TE_2.
2. Transférer les participations à des groupes de ro_2 à te_2.
3. Si TE_2 n'a qu'un attribut att_2
   a) min-card (te_2) = min-card (ro_1) * min-card (att_2),[3]
   a) max-card (te_2) = max-card (ro_1) * max-card (att_2),
   b) si card (ro_2) = 1-1 et ro_1 n'appartient pas à des groupes identifiant de TE_2 avec att_2, alors te_2 est identifiant de TE_1.
   c) transférer s'il y a lieu les attributs de att_2 dans et_2.

   Si TE_2 a plusieurs attributs att_{21},..., att_{2m}
   a) transférer att_{21},..., att_{2m} dans te_2,
   b) card (te_2) = card (ro_1),
   c) Si card (ro_2) = 1-1
      - les identifiants de TE_2 composés uniquement d'un sous-ensemble de att_{21},..., att_{2m}, identifie maintenant TE_1,
      - Si ro_1 n'appartient pas à des groupes identifiant de TE2 avec att_{21},..., att_{2m}, alors te_2 est identifiant de TE_1
4. Supprimer TA et ses rôles.
5. Supprimer TE_2 et ses attributs.

[3] min-card(x) * min-card(y) désigne le produit entre les cardinalités minimum des objets x et y. Si une des deux cardinalités vaut n alors le produit vaut n.. (idem pour max-card(x) * max-card(y))

3.5 Transformation d'un type d'association en groupe de référence

3.5.1 Description

Un type d'association 1-N ou 1-1 entre 2 types d'entités est remplacé par un groupe de référence, composé d'attributs. Cette transformation est utilisée pour éliminer le concept de type d'association dans tout le schéma, lorsque le SGBD cible ne le fournit pas.

3.5.2 Exemple

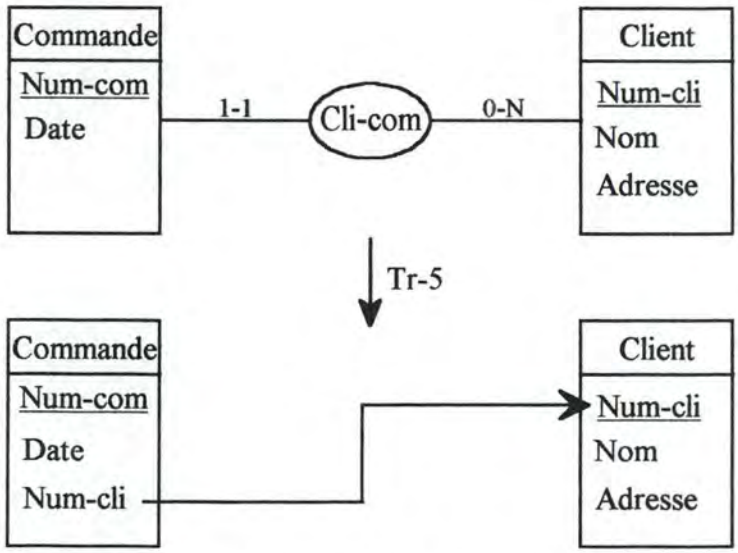


Figure 3.11 : Exemple de transformation d'un type d'association en groupe de référence

3.5.3 Définition

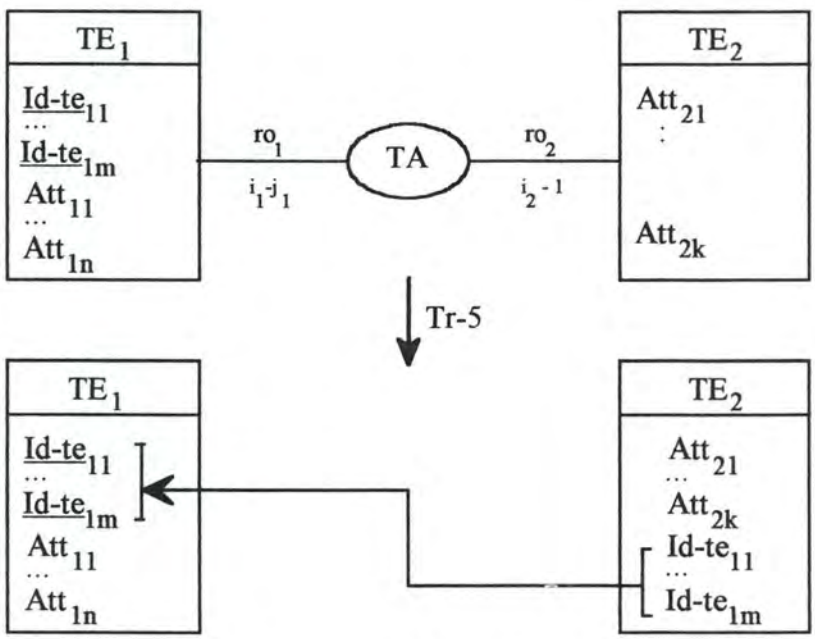


Figure 3.12 : Transformation d'un type d'association en attribut

Préconditions :

- Le type d'association TA est binaire sans attributs,
- Les rôles ro_1 et ro_2 sont mono-domaines,
- max-card (ro_2) = 1,
- Le type d'entité TE_1 a un groupe identifiant composé uniquement d'attributs.

Actions :

1. Créer pour chaque attributs id-te_{1k} ($1 \leq k \leq m$) composants du groupe identifiant de TE_1 une copie dans TE_2,
   a) min-rep (id-te_{1k} (TE_2)) = i_2,
   b) max-rep (id-te_{1k} (TE_2)) = 1.
2. Créer un groupe de référence dans TE_2 composé des attributs id-te_{1k} de TE_2.
3. Créer une contrainte d'inclusion du groupe de référence de TE_2 vers le groupes identifiant de TE_1. Si i_1 = 1, cette contrainte est une contrainte d'égalité.
4. Remplacer les participations à des groupes de ro_1 par id-te_{11},... id-te_{1m}.
5. Si j_1 = 1 alors id-te_{11},... id-te_{1m} est identifiant de TE_2.
6. Supprimer TA.

3.6 Transformation d'un groupe de référence en type d'association

3.6.1 Description

Un groupe composé d'un ou plusieurs attributs d'un type d'entité et qui référence un autre type d'entité (il y a une contrainte d'inclusion entre ce groupe et le groupe identifiant du type d'entité) est remplacé par un type d'association reliant ces deux types d'entité.

Cette transformation est la transformation inverse de la transformation présentée au §3.5.

3.6.2 Exemple

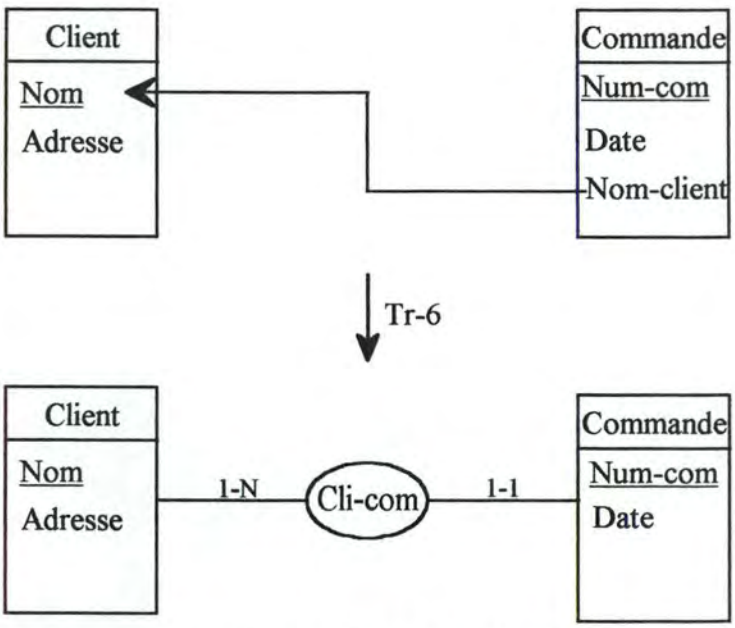


Figure 3.13 : Exemple de transformation d'un groupe de référence en type d'association

3.6.3 Définition

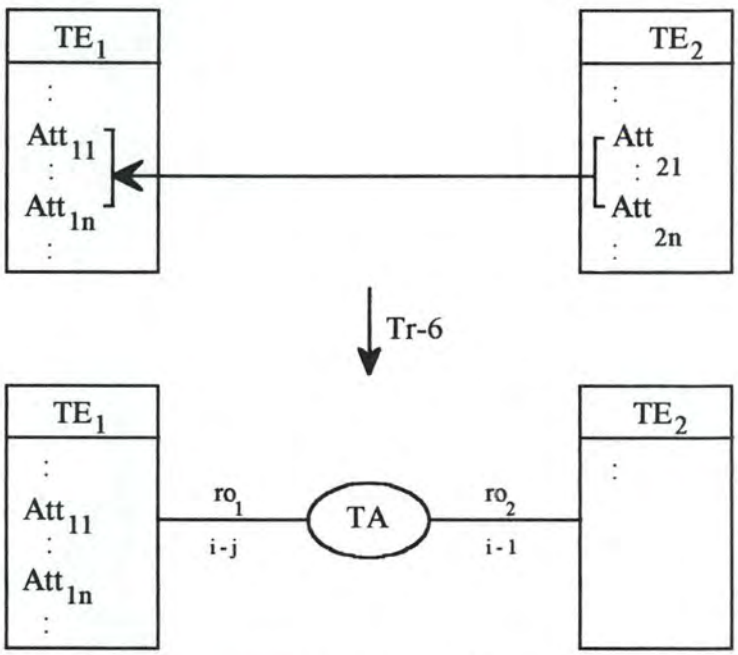


Figure 3.14 : Transformation d'un groupe de référence en type d'association

Préconditions :

- Il existe une contrainte d'inclusion entre le groupe Att_{21},... Att_{2n} et le groupe identifiant de TE_1,

- Le nombre de composants du groupe Att_{21},... Att_{2n} est le même que le nombre de composants de l'identifiant de TE_1,
- Le groupe identifiant de TE_1 n'est composé que d'attributs de TE_1,
- La longueur de la concaténation des composants du groupe Att_{21},... Att_{2n} est la même que la longueur de la concaténation des composants du groupe identifiant de TE_1,
- Les attributs Att_{21},... Att_{2n} ont tous une cardinalité 0-1 ou 1-1.

Actions :

1. Créer un type d'association TA entre TE_1 et TE_2 :
   - le nom de TA est la concaténation des nom courts de TE_1 et TE_2.
2. Si la contrainte entre le groupe Att_{21},... Att_{2n} et le groupe identifiant de TE_1 est une contrainte d'égalité,

   Alors min-card (ro_1) = 1,

   Sinon min-card(ro_2) = 0,
3. Si le groupe Att_{21},... Att_{2n} est identifiant de TE_2

   Alors max-card (ro_1) = 1,

   Sinon max-card (ro_2) = n,
4. card (ro_2) = card (Att_{21}),
5. Remplacer l'ensemble des attributs Att_{21},... Att_{2n} par ro_1 dans tous les groupes où ils apparaissent ensemble.
6. Supprimer le groupe et ses attributs Att_{21},... Att_{2n}.

3.7 Désagrégation d'un attribut décomposable

3.7.1 Description

Un attribut décomposable est remplacé par ses attributs composants, qui sont donc redéfinis à un niveau supérieur.

Cette transformation est utilisée pour éliminer tous les attributs décomposables d'un schéma, lorsque le SGBD cible n'accepte pas cette structure.

3.7.2 Exemple

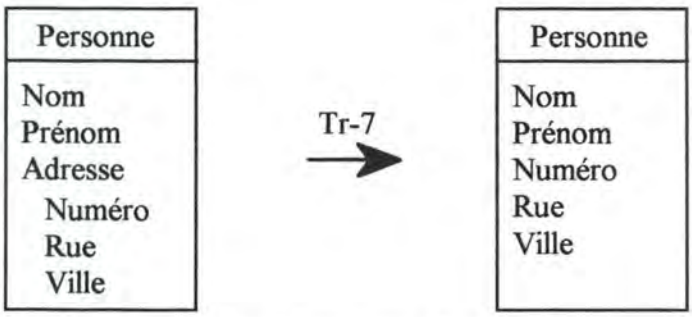


Figure 3.15 : Exemple de désagrégation d'un attribut décomposable

3.7.3 Définition

Le père d'un attribut peut être soit un type d'entité, soit un type d'association soit un attribut.

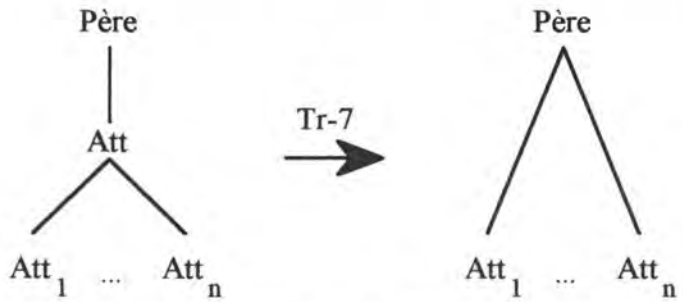


Figure 3.16 : Remplacement d'un attribut composé par ses composants

Précondition

- Att doit être un attribut décomposable,

- card (Att) = 1-1 ou $\forall$ i : card (Att_i) = 1-1 ($1 \leq i \leq n$).

Actions

1. Le père de Att devient le père des attributs Att_1,... Att_n,

2. $\forall$ i ($1 \leq i \leq n$) : min-card (Att_i) = min-card (Att) * min-card (Att_i),

   max-card (Att_i) = max-card(Att) * max-card (Att_i),

3. Créer un groupe composé de l'ensemble des attributs Att_1,... Att_n , et remplacer Att par ce groupe dans tous les groupes où il apparaît,

4. Supprimer Att.

3.8 Agrégation d'un ensemble d'attributs

3.8.1 Description

Un ensemble d'attribut est agrégé en un nouvel attribut composé. Cette transformation est la transformation inverse de la transformation présentée au §3.7.

3.8.2 Exemple

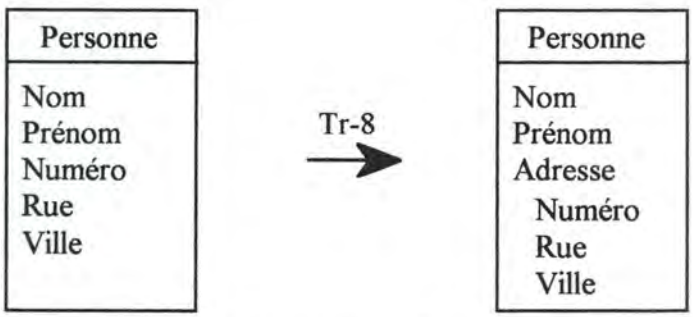


Figure 3.17 : Exemple d'agrégation d'un ensemble d'attributs

3.8.3 Définition

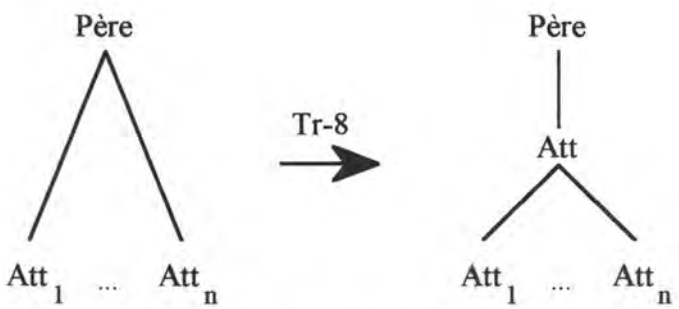


Figure 3.18 : Agrégation d'un ensemble d'attribut

Précondition :

- Tous les attributs de l'ensemble Att_1,... Att_n ont le même père.

Actions :

1. Créer un attribut décomposable Att de cardinalité 1-1 et dont le père est le même que celui des attributs Att_1,... Att_n,
2. Le père des attributs Att_1,... Att_n devient Att,
4. Remplacer les groupes composée uniquement de l'ensemble des attributs Att_1,... Att_n par Att dans tous les groupes où ils apparaissent.
5. Remplacer les attributs Att_1,... Att_n par Att dans tous les groupes où ils apparaisent ensemble.

4. Transformation vers le modèle relationnel

4.1. Le modèle relationnel

Le modèle relationnel est un des modèles de représentation d'informations les plus utilisés à l'heure actuelle. Il est basé essentiellement sur la représentation des informations sous forme de tables (ou relations). Beaucoup de SGBD relationnels existent actuellement sur le marché, on peut citer entre autre : DB2, ORACLE, INGRES, SQL/DS, RDB,...etc.

Cette section décrit le modèle et définit les concepts de relation, de domaine, d'attributs, de clé primaire et de clé étrangère. Le lecteur intéressé pourra consulter [DATE-90] pour plus de détails.

4.1.1 Les éléments de base du modèle

Les éléments de base du modèle relationnel sont les domaines et les relations :

Les domaines

Les domaines sont des ensembles de valeurs de même type (Ex : Caractère, entier, réel,...). Chaque domaine a un nom qui l'identifie; Ce nom est généralement choisi en rapport avec l'appellation usuelle des objets et propriétés désignés.

Les relations

Les relations sont des ensembles d'associations entre valeurs de domaines. Les associations (lignes ou n-uplets) d'une même relation ont la même structure. Un ligne de relation représente une association entre les constituants du réel perçu désignés par ces valeurs. Un domaine est utilisé dans une relation en tant qu'attribut de celle-ci; un domaine peut intervenir plus d'une fois dans une relation, mais chaque fois sous la forme d'attributs différents.

Formellement, une relation R définie sur un ensemble de domaines D_1, D_2,... D_n (non nécessairement distincts) est constituée de 2 parties, un *schéma* et une *extension* :

- Le *schéma* est constitué d'un ensemble fixe de paires attribut-domaine,

  $R(A_1 : D_1, A_2 : D_2,... A_n : D_n)$

  tel que chaque attribut A_j correspond à exactement un domaine D_j.

  Exemple : VOYAGE (Voyageur : CARACTERE, Départ : DATE, Retour : DATE)

- L'*extension* est constitué d'un ensemble variable de n-uplets (ou tuples) où chaque tuple est constitué d'un ensemble de paires attribut-valeur,

$(A_1 : v_{i1}, A_2 : v_{i2},... A_n : v_{in})$, i = 1,... m où m est le nombre de n-uplets de l'ensemble.

Exemple : (Voyageur : Dupont, Départ : 1/7/1992, Retour : 31/7/1992)

(Voyageur : Durand, Départ : 15/7/1992, Retour : 31/7/1992)

Une relation peut également être représentée sous la forme d'un tableau dont les colonnes correspondent aux attributs (Figure 3.19).

VOYAGEUR	DEPART	RETOUR
Dupont	1/7/1992	31/7/1992
Durand	15/7/1992	31/7/1992

Figure 3.19 : La relation VOYAGE

Il peut être utile d'enregistrer, au sujet d'un fait du réel perçu, un ensemble incomplet d'informations, certaines d'entre elles étant connues, d'autres restant encore à déterminer. En terme du modèle, ceci revient à construire un n-uplets dont certaines valeurs d'attributs sont encore inconnues. Nous admettons que de tels attributs puissent prendre une valeur particulière dite valeur "*nulle*".

4.1.2 Les contraintes d'intégrités

Une contrainte d'intégrité est une propriété formelle que les données doivent vérifier à tout instant. Ces propriétés sont choisies de manière à décrire le plus précisément le monde réel.

Nous allons présenter deux notions illustrant des contraintes d'intégrité du modèle relationnel : les clés primaires et les clés étrangères.

Les clés primaires

L'identifiant d'un relation est un attribut ou une liste d'attributs telle qu'il ne peut se trouver dans cette relation deux lignes qui possèdent les mêmes valeurs pour les attributs de cette liste.

Par exemple (figure 3.20), l'attribut NUM-CLI est identifiant de la relation CLIENT.

NUM-CLI	NOM	VILLE
C110	Dupont	Namur
B221	Durand	Andenne
B062	Dupont	Andenne
C734	Dubois	Namur

Figure 3.20 : La relation CLIENT

Dans le modèle, une clé primaire désigne un identifiant d'une relation. Comme une relation peut avoir plusieurs identifiant mais ne peut avoir qu'une seule clé primaire, on désigne par clés secondaires les autres identifants de la relation.

La règle d'intégrité d'entité impose qu'aucun composant de la clé primaire d'une relation ne peut prendre de valeur nulle.

Les clés étrangères

Une clé étrangère est un attribut ou une liste d'attributs d'une relation R1 dont les valeurs sont prises parmi les valeurs d'une clé primaire d'une autre relation R2 (R1 et R2 non nécessairement distinctes). Une valeur de clé étrangère représente une référence au n-uplet dont la valeur de clé primaire est la même. Par exemple, l'attribut NUM-CLI de la relation COMMANDE (figure 3.21) est une référence vers la clé primaire NUM-CLI de la relation CLIENT(figure 3.20).

NUM-COM	NUM-CLI	DATE
151	C110	1/7/92
152	B062	23/7/92
153	C110	25/7/92
154	C734	31/7/92

Figure 3.21 : La relation COMMANDE

La contrainte d'intégrité référentielle impose que l'ensemble des valeurs d'une clé étrangère soit inclus dans l'ensemble de valeurs de la clé primaire référencée. Cela signifie simplement que *si B référence A, alors A doit exister*.

4.2. Le processus de transformation

Cette section décrit dans un premier temps l'ensemble des règles que le modèle de spécification implémenté doit respecter pour être conforme au modèle relationnel. Ensuite, le processus de transformation du modèle Entité/Association étendu vers le modèle relationnel sera décrit.

4.2.1 Les règles de conformité

Le modèle de spécification permet de décrire une base de données selon le modèle relationnel. Il est évident que pour que la description soit conforme au modèle relationnel, il faut imposer certaines restrictions dans la spécification. Ces restrictions portent sur tous les objets du modèle de spécification.

Les principales restrictions des structures relationnelles par rapport à celle du modèle de spécification sont les suiviantes :

- Il n'y a pas de types d'associations et de rôles,
- Tous les attributs sont simples (non répétitifs) et élémentaires (non décomposables),
- Il y a au moins un attribut par type d'entité,
- Tous les groupes sont composés d'attributs locaux,
- Il n'y a pas de contraintes d'intégrité autres que les contraintes d'intégrité référentielle entre groupes.

Un description qui respecte l'ensemble de ces règles est dites conforme au modèle relationnel.

4.2.2 Transformation vers le modèle relationnel

La transformation du modèle de spécification vers le modèle relationnel se fait en 3 étapes :

1. Transformation des types d'associations complexes et des types d'association N-N[4] en types d'entités.
2. Tant que le schéma contient des attributs répétitifs ou des attributs décomposables :
   a) Transformation des attributs répétitifs en types d'entités,
   b) Désagrégation des attributs décomposables.
3. Transformation des types d'associations fonctionnels en attributs de référence.

[4] Les types d'association N-N sont de degré 2, sans attributs et avec 2 rôles de cardinalités i-n.

Chapitre 4 : Les outils de génération

Introduction

Les outils de génération sont utilisés dans la dernière phase de conception de la base de données (conception physique). A partir du modèle du SGBD cible choisi (relationnel, réseau,...) et des régles définies sur ce SGBD, on peut établir une liste de contraintes que doit respecter le schéma de la base de données pour être conforme au SGBD. Une fois la conformité établie, la génération peut avoir lieu.

Le SGBD cible choisi dans le cadre de ce travail est le SGBD relationnel DB2.

La première section de ce chapitre présente **le SGBD relationnel DB2**, elle introduit ses principaux concepts et décrit la syntaxe du langage.

La deuxième section, **Correspondance DB2 - Base des spécifications**, énonce les règles de conformité et établit la correspondance entre les concepts.

1. Le SGBD relationnel DB2

Cette section présente le SGBD relationnel DB2 et décrit sa syntaxe. Les concepts présentés sont uniquement ceux qui nous intéressent dans le cadre de la génération de code exécutable à partir de la base des spécifications. Le lecteur intéressé par une description complète de DB2 peut consulter [DB2-89].

1.1. Concepts de base

DB2 est un système de gestion de base de données qui suit la norme ANSI SQL [ANSI-86] définie par l'American National Standards Institute. SQL (Structured Query Langage) est le langage utilisé pour accéder aux données d'une base de données relationnelle. Il fournit la capacité de définir et de manipuler des données. Il peut être utilisé pour définir des objets tels que des tables ou des index. Il permet également de retrouver, d'insérer, de mettre à jour et d'effacer des données, et de gérer les autorisations d'accès aux données.

1.1.1 Les tables

Une base de données relationnelle est un ensemble de tables. Les tables sont composées de lignes et de colonnes. L'intersection d'une ligne et d'une colonne, constitue une donnée élémentaire appelée valeur. Une colonne est un ensemble de valeurs du même type. Une ligne est une suite de valeurs tel que la $n^{ième}$ valeur est une valeur de la $n^{ième}$ colonne de la table. Une table est définie par l'instruction CREATE TABLE.

1.1.2 Les index

Un index est un ensemble ordonné de pointeurs vers les lignes d'une table. Chaque index est basé sur les valeurs des données d'une ou plusieurs colonnes d'une table. Un index est un objet qui est séparé des données dans la table. Lorsqu'on demande un index, le SGBD construit et gère cette structure automatiquement.

Les index sont utilisés pour :

- Améliorer les performances. Dans la plupart des cas l'accès aux données est plus rapide que sans index.
- Assurer l'unicité. Une table avec un index unique ne peut avoir plusieurs rangées avec des valeurs de clés identiques. (Une clé est une colonne, ou un ensemble ordonné de colonnes sur lesquelles l'index est créé.)

1.2 La syntaxe DB2

1.2.1 La création de la base de données

L'instruction CREATE DATABASE définit une base de données dans laquelle des tables et des index seront définis par la suite.

Syntaxe :

DB-def ::= **create database** database-name**;**

Description :

database-name

est le nom de la base de données

1.2.2 La création des tables

L'instruction CREATE TABLE définit une table. La définition doit inclure son nom et les noms de ses colonnes.

Syntaxe :

TABLE-def ::= **create table** table-name **(** COLUMN-def [**,** COLUMN-def]...
[**,** PRIMARY-KEY-def]
[**,** FOREIGN-KEY-def [**,** FOREIGN-KEY-def]...]**);**

COLUMN-def ::= column-name data-type [**not null**]

PRIMARY-KEY-def ::= **primary key (** column-name [**,** column-name]...**)**

FOREIGN-KEY-def ::= **foreign key (** column-name [**,** column-name]...**)**
references table-name

Description :

table-name

est le nom de la table, ce nom doit être unique dans la base de données.

column-name

est le nom d'une colonne d'une table, ce nom doit être unique dans la table.

data-type

est une des types de la liste suivante :

decimal (entier,entier)

pour un nombre décimal. Le premier entier est la précision du nombre (nombre total de chiffres), il peut varier de 1 à 15. Le second représente le nombre de chiffres à droite de la virgule, il peut varier de 0 à la précision. On peut également spécifier :

decimal (entier) pour **decimal** (entier,**0**)

char(entier)

pour les chaînes de caractères de longueur fixe. L'entier représente la longueur de la chaîne et peut varier de 1 à 254.

varchar(entier)

pour les chaînes de caractères de longueur variable. L'entier représente la longueur maximale de la chaîne.

date

pour les dates.

La clause ***not null***

empêche une colonne de contenir des valeurs nulles.

La clause ***primary key***

définit une clé primaire composée des colonnes citées. La clause ne peut pas être spécifiée plus d'une fois et les colonnes citées doivent être définies avec la clause ***not null***. Chaque *column-name* doit identifier une colonne de la table et la même colonne ne peut pas apparaître plus d'une fois. Le nombre de colonnes ne peut pas dépasser 16 et la somme de leur longueur ne peut pas dépasser 254.

La clause ***foreign key***

définit une clé étrangère composée des colonnes citées. Chaque clause *foreign key* définit une contrainte référentielle. Chaque *column-name* doit identifier une colonne de la table, et la même colonne ne peut pas apparaître plus d'une fois. Le nombre de colonnes ne peut pas dépasser 16, et la somme de la longueur de leurs attributs ne peut pas être supérieur à 254 moins le nombre de colonnes où la valeur nulle est autorisée. Le *table-name* de la clause identifie une table déjà définie. Cette table doit avoir une clé primaire qui a le même nombre de colonnes que la clé étrangère et, excepté pour leur nom et la clause null, la description de la $n^{ième}$ colonne de cette clé primaire doit être identique à la description de la $n^{ième}$ colonne de la clé étrangère.

1.2.3 La création des index

L'instruction CREATE INDEX crée un index sur une table

Syntaxe :

INDEX-def ::= **create [unique] index** index-name **on** table-name

(column-name [,column-name]...**);**

Description :

la clause ***unique***

empêche la table de contenir deux ou plusieurs lignes avec la même valeur de la clé d'index. La contrainte est vérifiée lorsque des lignes de la table sont modifiées et lorsque de nouvelles lignes sont insérées.

Lorsque la clause unique est utilisée, les valeurs nulles sont traitées comme les autres valeurs. Par exemple, si la clé d'index est composée d'une seule colonne qui peut contenir des valeurs nulles, alors cette colonne ne peut contenir qu'une seule valeur nulle.

index-name

est le nom de l'index, il doit être unique dans la base de données

table-name

est le nom de la table sur laquelle porte l'index. Cette table doit déjà avoir été définie.

column-name

sont les noms des colonnes faisant partie de la clé d'index. Chacun de ces noms identifie une colonne de la table. Le nombre de colonnes ne peut pas dépasser 16 et la même colonne ne peut pas être spécifié plus d'une fois. La somme de la longueur des attributs des colonnes ne peut pas être supérieur à 254-N, où N représente le nombre de colonnes pouvant contenir des valeurs nulles.

2. La correspondance DB2-Base des spécifications

Cette section énonce dans un premier temps les règles de conformité à un schéma DB2. On décrit ensuite le schéma E/A du SGBD DB2[1] et on établit une correspondance avec le schéma de la base des spécifications.

2.1 Les règles de conformité

Les règles définies ci-dessous définissent des contraintes que doit respecter la description d'une base de donnée pour pouvoir produire un schéma exécutable par le système de gestion de base de données DB2. Les règles marquées d'une * sont des règles que doit respecter tout schéma pour être conforme au modèle relationnel.

Contraintes sur les types d'entités :

- Il y a au moins un attribut par type d'entité.*
- Un type d'entité contient au plus 300 attributs.

Contraintes sur les types d'associations :

- Il n'y a pas de types d'associations.*

Contraintes sur les attributs :

- Tous les attributs sont simples et élémentaires.*
- Un attribut de type numérique ne peut avoir une longueur supérieure à 15 et un nombre de décimale supérieur à la longueur.

Contraintes sur les groupes :

- Tous les groupes sont composés d'attributs locaux.*
- Tous les groupes identifiants secondaires sont des clés d'accès.
- Un groupe ne peut pas avoir plus de 16 composants.
- Un groupe identifiant primaire ne peut pas contenirs d'attributs facultatifs.
- La somme de la longueur des composants d'un groupe ne peut pas dépasser 254-N, où N représente le nombre d'attributs facultatifs du groupe.

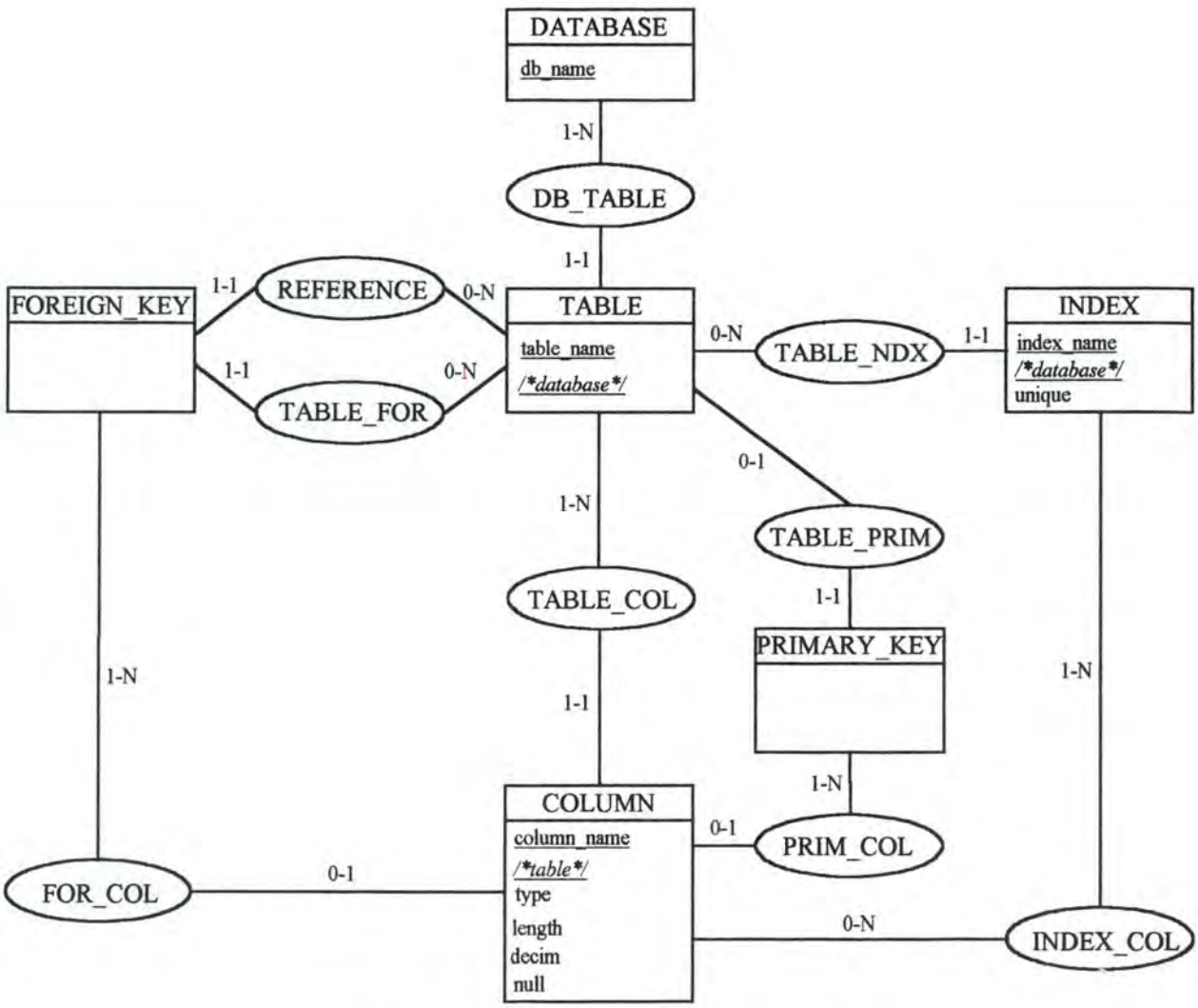
[1] Il s'agit d'un sous-schéma car tous les concepts ne sont pas représentés.

Contraintes sur les noms :

- Le nom d'un type d'entité ou d'un attribut est composé de 1 à 18 caractères et le nom d'un schéma de 1 à 8. Le premier caractère est une lettre, les autres sont des lettres, des chiffres ou le caractère "_".
- Un nom ne peut appartenir à la liste des mots réservés du SGBD DB2.

2.2 Schema E/A des concepts DB2

Les concepts des bases de données DB2 présentés à la section précédente peuvent être formalisé en un schéma Entité/Association. Ce schéma, représenté à la figure 4.1, nous permet de comprendre et d'analyser les relations qu'il y a entre ces objets.



Figute 4.1 : Schema E/A de la base de données DB2

Une base de données (**DATABASE**) est identifiée par son nom (***db_name***) et contient une ou plusieurs tables (**TABLE**).

Dans une base de données, une table (**TABLE**) est identifiée par son nom (***table_name***). Elle contient une ou plusieurs colonnes (**COLUMN**), zéro ou plusieurs index (**INDEX**), zéro ou plusieurs clés étrangères (**FOREIGN_KEY**) et éventuellement une clé primaire (**PRIMARY_KEY**).

Dans une table, une colonne (**COLUMN**) est identifiée par son nom (***column_name***) et est caractérisée par un type (***type***), une longueur (***length***) et un nombre de décimale (***decim***). L'attribut ***null*** indique si la colonne peut prendre des valeurs nulles.

Dans une base de données, un index (**INDEX**) est identifié par son nom (***index_name***). L'attribut ***unique*** indique si l'index est unique ou non. Un index porte sur une et une seule table (**TABLE**) et est défini sur une ou plusieurs colonnes (**COLUMN**) de cette table.

Une clé primaire (**PRIMARY_KEY**) peut être définie sur une table, elle est composée d'une ou plusieurs colonnes de cette table.

Des clés étrangères (**FOREIGN_KEY**) peuvent être définies sur une table via **TABLE_FOR**, ces clés sont composées d'une ou plusieurs colonnes de la table. Chaque clé étrangère référence une table via **REFERENCE**.

2.3 Les correspondances

A chaque objet du schéma E/A de la base de données DB2 on peut faire correspondre un ou plusieurs objets de la base des spécifications de l'atelier logiciel.

Le type d'entité **DATABASE** correspond au type d'entité **SCHEMA** et son attribut ***db_name*** à l'attribut ***name*** de **SCHEMA**.

Le type d'entité **TABLE** correspond au type d'entité **ENTITY_TYPE** et son attribut ***table_name*** à l'attribut ***name*** de **ENTITY_TYPE**.

Le type d'entité **INDEX** reprend les entités de **GROUP** qui sont clé d'accès, c'est-à-dire dont l'attribut ***key*** est VRAI. L'attribut ***index_name*** n'a pas d'équivalent dans le schéma de la base des spécifications, il est déterminé lors de la génération. L'attribut ***unique*** correspond à l'attribut ***identifier*** du type d'entité **GROUP**.

Le type d'entité **COLUMN** correspond au type d'entité **ATTRIBUTE** et son attribut ***column_name*** à l'attribut ***name*** de **ATTRIBUTE**. Les attributs ***type***, ***length*** et ***decim*** de **COLUMN** correspondent aux attributs ***type***, ***length*** et ***decim*** de **ATTRIBUTE**. L'attribut ***null*** est VRAI si l'attribut ***min_rep*** de **ATTRIBUTE** vaut 0.

Le type d'entité **PRIMARY_KEY** reprend les entités de **GROUP** qui sont identifiant primaire, c'est-à-dire dont les attribut ***identifier*** et ***status*** sont VRAI.

Le type d'entité **FOREIGN_KEY** reprend les entités de **GROUP** qui sont origines d'une référence, c'est-à-dire qu'une contrainte référentielle est définie sur ces groupes et qu'ils jouent le

rôle de *référençant* dans la contrainte. La table (**TABLE**) référencée correspond au type d'entité (**ENTITY_TYPE**) qui contient le groupe (**GROUP**) jouant le rôle de *référencé* dans la contrainte.

Tableau récapitulatif des correspondances :

Schéma DB2	Schéma de la base des spécifications
DATABASE ***db_name***	**SCHEMA** ***name*** (nom du schéma)
TABLE ***table_name***	**ENTITY_TYPE** (type d'entité) ***name*** (nom du type d'entité)
INDEX ***index_name*** ***unique***	**GROUP** (groupe clé d'accès) créé à la génération ***identifier*** (groupe identifiant)
COLUMN ***column_name*** ***type*** ***length*** ***decim*** ***null***	**ATTRIBUTE** (attribut) ***name*** (nom de l'attribut) ***type*** (type de l'attribut) ***length*** (longueur de l'attribut) ***decim*** (nombre de décimale de l'attribut) ***min_rep*** (cardinalité minimum de l'attribut)
PRIMARY_KEY	**GROUP** ***identifier*** + ***status*** (groupe identifiant primaire)
FOREIGN_KEY	**GROUP** (groupe référençant)

Conclusion

1. Apports de ce travail

Au terme de ce travail, nous pouvons dire que l'utilisation d'un langage orienté objet pour le développement d'un atelier de conception de bases de données constitue un avantage important grâce aux possibilités d'extensibilité et de réutilisabilité qu'apporte cette approche.

Le modèle E/A étendu utilisé pour décrire le schéma conceptuel de la base des spécifications constitue une technique idéale pour modéliser les structures de données à mémoriser dans une approche orientée objet.

2. Possibilités d'extensions du travail

La première extension à envisager pour ce travail est l'implémentation des concepts de Généralisation/Spécialisation, de domaines et de contraintes d'intégrité présenté au chapitre 2 mais non intégré à l'atelier. L'apport de nouveaux concepts tels que les descriptions sur les objets ou bien les statistiques serait également intéressant. Toutes ces nouvelles notions doivent être étudiée en tenant compte de leur influence sur les outils de transformation et de génération existants.

D'autre transformations sur le modèle de spécification peuvent également être implémentées. On peut citer par exemple les transformations sur les structures de généralisation/spécialisation ou sur les domaines, la transformation des rôles multi-domaines, la fusion et l'éclatement de types d'entités, les transfomations sur les groupes et sur les noms.

Des transformations vers d'autres modèles de données (CODASYL, COBOL,...) est également à envisager ainsi que l'extension à des générateurs vers d'autres SGBD relationnels.

Bibliographie

[BACN-92]
BATINI Carlo, CERI Stefano, NABATHE Shamkant B.,
Conceptual Database Design, An Entity-Relationship Approach,
The Benjamin/Cummings Publishing Company Inc., 1992.

[BOPI-89]
BODART François, PIGNEUR Yves,
Conception assistée des applications informatiques, Tome 1 : Méthodes - Modèles - Outils,
Masson, 1989.

[DATE-90]
DATE C. J.,
An introduction to Database Systems, Volume 1,
Addison-Wesley Publishing Company, Fifth Edition, 1990.

[DELO-91]
DELOBEL Claude,LECLUSE Christophe,RICHARD Philippe,
Bases de données : Des systèmes relationnels aux systèmes à objest,
InterEditions, 1991.

[HAIN-86]
HAINAUT Jean-Luc,
Conception assistée des applications informatiques : Tome 2 : Conception de la base de données,
Masson, Presses universitaires de Namur, 1986.

[MEYE-90]
MEYER Bertrand,
Conception et programmation par objets,
InterEditions, 1990.

[STRO-92]
STROUSTRUP Bjarne,
Le langage C++,
Editions Addison-Wesley France SA, 1992.

[DEMA-92]
DECUYPER Bernard, MARCHAND Olivier,
Description du meta-schéma dans une perspective orienté objet de l'atelier logiciel TRAMIS, Rapport technique,
Institut d'informatique, Namur, 1992.

[HAIN-89]
HAINAUT Jean-Luc,
Bases de données et bases de connaissances en gestion des organisations,
Cinquième Ecole d'Automne de Bases de Données, AFCET, Novembre 1989.

[HAIN-91]
HAINAUT Jean-Luc,
A Temporal Statistical Model for Entity-Relationship Schemas,
Institut d'Informatique - University of Namur, 1991.

[HCDM-91]
HAINAUT Jean-Luc, CADELLI Mario, DECUYPER Bernard, MARCHAND Olivier,
Database CASE Tool Architecture : Principles for flexible design architecture,
Institut d'Informatique - University of Namur, 1991.

[ROST-91]
ROBERFROID Eva-Marie, STEINIER Stephane,
Conception et réalisation d'un environnement de développement d'applications bureautiques,
Mémoire de Licence et Maîtrise, Institut d'Informatique, F.N.D.P., Namur, 1991.

[ANSI-86]
Database Language SQL,
final draft ISO 9075-1987(E), (ANSI X3H2 report) November 1986.

[DB2-89]
IBM DATABASE 2 version 2, SQL reference Release 2,
IBM documentation, SC26-4380-1, September 1989